Roman Wyrzykowski
Jack Dongarra
Konrad Karczewski
Jerzy Waśniewski (Eds.)

LNCS 8385

Parallel Processing and Applied Mathematics

10th International Conference, PPAM 2013
Warsaw, Poland, September 8–11, 2013
Revised Selected Papers, Part II

2 Part II

Springer

Lecture Notes in Computer Science 8385

Roman Wyrzykowski · Jack Dongarra
Konrad Karczewski · Jerzy Waśniewski (Eds.)

Parallel Processing and Applied Mathematics

10th International Conference, PPAM 2013
Warsaw, Poland, September 8–11, 2013
Revised Selected Papers, Part II

Springer

Editors
Roman Wyrzykowski
Konrad Karczewski
Institute of Computer and
Information Science
Czestochowa University of Technology
Czestochowa
Poland

Jerzy Waśniewski
Informatics and Mathematical Modelling
Technical University of Denmark
Kongens Lyngby
Denmark

Jack Dongarra
Department of Computer Science
University of Tennessee
Knoxville, TN
USA

ISSN 0302-9743 ISSN 1611-3349 (electronic)
ISBN 978-3-642-55194-9 ISBN 978-3-642-55195-6 (eBook)
DOI 10.1007/978-3-642-55195-6
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014937670

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

Preface

This volume comprises the proceedings of the 10th International Conference on Parallel Processing and Applied Mathematics, PPAM 2013, which was held in Warsaw, Poland, September 8–11, 2013. The jubilee PPAM conference was organized by the Department of Computer and Information Science of the Czestochowa University of Technology, under the patronage of the Committee of Informatics of the Polish Academy of Sciences, in cooperation with the Polish-Japanese Institute of Information Technology. The main organizer was Roman Wyrzykowski.

PPAM is a biennial conference. Nine previous events have been held in different places in Poland since 1994. The proceedings of the last six conferences have been published by Springer-Verlag in the *Lecture Notes in Computer Science* series (Nałęczów, 2001, vol. 2328; Częstochowa, 2003, vol. 3019; Poznań, 2005, vol. 3911; Gdańsk, 2007, vol. 4967; Wrocław, 2009, vols. 6067 and 6068; Toruń, 2011, vols. 7203 and 7204).

The PPAM conferences have become an international forum for exchanging ideas between researchers involved in parallel and distributed computing, including theory and applications, as well as applied and computational mathematics. The focus of PPAM 2013 was on models, algorithms, and software tools that facilitate efficient and convenient utilization of modern parallel and distributed computing architectures, as well as on large-scale applications.

This meeting gathered the largest number of participants in the history of PPAM conferences – more than 230 participants from 32 countries. A strict refereeing process resulted in the acceptance of 143 contributed presentations, while approximately 44 % of the submissions were rejected. Regular tracks of the conference covered such important fields of parallel/distributed/cloud computing and applied mathematics as:

– Numerical algorithms and parallel scientific computing
– Parallel non-numerical algorithms
– Tools and environments for parallel/distributed/cloud computing
– Applications of parallel computing
– Applied mathematics, evolutionary computing, and metaheuristics

The plenary and invited talks were presented by:

– Fran Berman from the Rensselaer Polytechnic Institute (USA)
– Ewa Deelman from the University of Southern California (USA)
– Jack Dongarra from the University of Tennessee and Oak Ridge National Laboratory (USA), and University of Manchester (UK)
– Geoffrey Ch. Fox from Indiana University (USA)
– Laura Grigori from Inria (France)
– Fred Gustavson from the IBM T.J. Watson Research Center (USA)
– Georg Hager from the University of Erlangen-Nuremberg (Germany)
– Alexey Lastovetsky from the University College Dublin (Ireland)

– Miron Livny from the University of Wisconsin (USA)
– Piotr Luszczek from the University of Tennessee (USA)
– Rizos Sakellariou from the University of Manchester (UK)
– James Sexton from the IBM T.J. Watson Research Center (USA)
– Leonel Sousa from the Technical University of Lisbon (Portugal)
– Denis Trystram from the Grenoble Institute of Technology (France)
– Jeffrey Vetter from the Oak Ridge National Laboratory and Georgia Institute of Technology (USA)
– Richard W. Vuduc from the Georgia Institute of Technology (USA)
– Robert Wisniewski from Intel (USA)

Important and integral parts of the PPAM 2013 conference were the workshops:

– Minisympsium on GPU Computing organized by José R. Herrero from the Universitat Politecnica de Catalunya (Spain), Enrique S. Quintana-Ortí from the Universidad Jaime I (Spain), and Robert Strzodka from NVIDIA
– Special Session on Multicore Systems organized by Ozcan Ozturk from Bilkent University (Turkey), and Suleyman Tosun from Ankara University (Turkey)
– Workshop on Numerical Algorithms on Hybrid Architectures organized by Przemysław Stpiczyński from the Maria Curie Skłodowska University (Poland), and Jerzy Waśniewski from the Technical University of Denmark
– Workshop on Models, Algorithms and Methodologies for Hierarchical Parallelism in New HPC Systems organized by Giulliano Laccetti and Marco Lapegna from the University of Naples Federico II (Italy), and Raffaele Montella from the University of Naples Parthenope (Italy)
– Workshop on Power and Energy Aspects of Computation organized by Richard W. Vuduc from the Georgia Institute of Technology (USA), Piotr Luszczek from the University of Tennessee (USA), and Leonel Sousa from the Technical University of Lisbon (Portugal)
– Workshop on Scheduling for Parallel Computing, SPC 2013, organized by Maciej Drozdowski from Poznań University of Technology (Poland)
– The 5th Workshop on Language-Based Parallel Programming Models, WLPP 2013, organized by Ami Marowka from the Bar-Ilan University (Israel)
– The 4th Workshop on Performance Evaluation of Parallel Applications on Large-Scale Systems organized by Jan Kwiatkowski from Wrocław University of Technology (Poland)
– Workshop on Parallel Computational Biology, PBC 2013, organized by David A. Bader from the Georgia Institute of Technology (USA), Jarosław Żola from Rutgers University (USA), and Bertil Schmidt from the University of Mainz (Germany)
– Minisymposium on Applications of Parallel Computations in Industry and Engineering organized by Raimondas Čiegis from Vilnius Gediminas Technical University (Lithuania), and Julius Žilinskas from Vilnius University (Lithuania)
– Minisymposium on HPC Applications in Physical Sciences organized by Grzegorz Kamieniarz and Wojciech Florek from A. Mickiewicz University in Poznań (Poland)

– Minisymposium on Applied High-Performance Numerical Algorithms in PDEs organized by Piotr Krzyżanowski and Leszek Marcinkowski from Warsaw University (Poland), and Talal Rahman from Bergen University College (Norway)
– Minisymposium on High-Performance Computing Interval Methods organized by Bartłomiej J. Kubica from Warsaw University of Technology (Poland)
– Workshop on Complex Colective Systems organized by Paweł Topa and Jarosław Wąs from AGH University of Science and Technology in Kraków (Poland)

The PPAM 2013 meeting began with five tutorials:

– Scientific Computing on GPUs, by Dominik Göddeke from the University of Dortmund (Germany), and Robert Strzodka from NVIDIA
– Design and Implementation of Parallel Algorithms for Highly Heterogeneous HPC Platforms, by Alexey Lastovetsky from University College Dublin (Ireland)
– Node Level Performance Engineering, by Georg Hager from the University of Erlangen-Nuremberg (Germany)
– Delivering the OpenCl Performance Promise: Creating and Optimizing OpenCl Applications with the Intel OpenCl SDK, by Maxim Shevtsov from Intel (Russia)
– A History of A Central Result of Linear Algebra and the Role of that Gauss, Cholesky and Others Played in Its Development, by Fred Gustavson from the IBM T.J. Watson Research Center (USA)

The PPAM Best Poster Award is granted to the best poster on display at the PPAM conferences, and was established at PPAM 2009. This award is bestowed by the Program Committee members to the presenting author(s) of the best poster. The selection criteria are based on the scientific content, and on the quality of the poster presentation. The PPAM 2013 winners were Lars Karlsson, and Carl Christian K. Mikkelsen from Umea University, who presented the poster "Improving Perfect Parallelism." The Special Award was bestowed to Lukasz Szustak, and Krzysztof Rojek from the Częstochowa University of Technology, and Pawel Gepner from Intel, who presented the poster "Using Intel Xeon Phi to Accelerate Computation in MPDATA Algorithm."

A new topic was introduced at PPAM 2013: *Power and Energy Aspects of Computation (PEAC)*. Recent advances in computer hardware rendered the issues related to power and energy consumption as the driving metric for the design of computational platforms for years to come. Power-conscious designs, including multicore CPUs and various accelerators, dominate large supercomputing installations as well as large industrial complexes devoted to cloud computing and the big data analytics. At stake are serious financial and environmental impacts, which the large-scale computing community has to now consider and embark on careful re-engineering of software to fit the demanding power caps and tight energy budgets.

The workshop presented research into new ways of addressing these pressing issues of energy preservation, power consumption, and heat dissipation while attaining the best possible performance levels at the scale demanded by modern scientific challenges.

The PEAC Workshop, as well as the conference as a whole, featured a number of invited and contributed talks covering a diverse array of recent advances, including:

- Cache-aware roofline model for monitoring performance and power in connection with application characterization (by L. Sousa et al.)
- Resource scheduling and allocation schemes based on stochastic models (by M. Oxley et al.)
- A comprehensive study of iterative solvers on a large variety of computing platforms including modern CPUs, accelerators, and embedded computers (by Enrique S. Quintana-Ortí et al.)
- Energy and power consumption trends in HPC (by P. Luszczek)
- Sensitivity of graph metrics to missing data and the benefits they have for overall energy consumption (by A. Zakrzewska et al.)
- Cache energy models and their analytical properties in the context of embedded devices (by K. de Vogeleer et al.)
- Predictive models for execution time, energy consumption, and power draw of algorithms (by R. Vuduc)

The organizers are indebted to the PPAM 2013 sponsors, whose support was vital to the success of the conference. The main sponsor was the Intel Corporation. The other sponsors were: IBM Corporation, Hewlett-Packard Company, Rogue Wave Software, and AMD. We thank to all the members of the international Program Committee and additional reviewers for their diligent work in refereeing the submitted papers. Finally, we thank all of the local organizers from the Częstochowa University of Technology, and the Polish-Japanese Institute of Information Technology in Warsaw, who helped us to run the event very smoothly. We are especially indebted to Grażyna Kołakowska, Urszula Kroczewska, Łukasz Kuczyński, Adam Tomaś, and Marcin Woźniak from the Częstochowa University of Technology; and to Jerzy P. Nowacki, Marek Tudruj, Jan Jedliński, and Adam Smyk from the Polish-Japanese Institute of Information Technology.

We hope that this volume will be useful to you. We would like everyone who reads it to feel invited to the next conference, PPAM 2015, which will be held September 6–9, 2015, in Kraków, the old capital of Poland.

January 2014

Roman Wyrzykowski
Jack Dongarra
Konrad Karczewski
Jerzy Waśniewski

Organization

Program Committee

Jan Węglarz	Poznań University of Technology, Poland (Honorary Chair)
Roman Wyrzykowski	Częstochowa University of Technology, Poland (Program Committee Chair)
Ewa Deelman	University of Southern California, USA (Program Committee Vice-Chair)
Francisco Almeida	Universidad de La Laguna, Spain
Pedro Alonso	Universidad Politecnica de Valencia, Spain
Peter Arbenz	ETH, Zurich, Switzerland
Piotr Bała	Nicolaus Copernicus University, Poland
David A. Bader	Georgia Institute of Technology, USA
Michael Bader	TU München, Germany
Włodzimierz Bielecki	West Pomeranian University of Technology, Poland
Paolo Bientinesi	RWTH Aachen, Germany
Radim Blaheta	Institute of Geonics, Czech Academy of Sciences
Jacek Błażewicz	Poznań University of Technology, Poland
Adam Bokota	Częstochowa University of Technology, Poland
Pascal Bouvry	University of Luxembourg
Tadeusz Burczyński	Silesia University of Technology, Poland
Jerzy Brzeziński	Poznań University of Technology, Poland
Marian Bubak	AGH Kraków, Poland, and University of Amsterdam, The Netherlands
Christopher Carothers	Rensselaer Polytechnic Institute, USA
Jesus Carretero	Universidad Carlos III de Madrid, Spain
Raimondas Čiegis	Vilnius Gediminas Technical University, Lithuania
Andrea Clematis	IMATI-CNR, Italy
Jose Cunha	University Nova of Lisbon, Portugal
Zbigniew Czech	Silesia University of Technology, Poland
Jack Dongarra	University of Tennessee and ORNL, USA, and University of Manchester, UK
Maciej Drozdowski	Poznań University of Technology, Poland
Erik Elmroth	Umea University, Sweden
Mariusz Flasiński	Jagiellonian University, Poland
Franz Franchetti	Carnegie Mellon University, USA
Tomas Fryza	Brno University of Technology, Czech Republic
Pawel Gepner	Intel Corporation

Domingo Gimenez	University of Murcia, Spain
Mathieu Giraud	LIFL and Inria, France
Jacek Gondzio	University of Edinburgh, UK
Andrzej Gościński	Deakin University, Australia
Laura Grigori	Inria, France
Adam Grzech	Wroclaw University of Technology, Poland
Inge Gutheil	Forschungszentrum Juelich, Germany
Georg Hager	University of Erlangen-Nuremberg, Germany
José R. Herrero	Universitat Politecnica de Catalunya, Barcelona, Spain
Ladislav Hluchy	Slovak Academy of Sciences, Slovakia
Florin Isaila	Universidad Carlos III de Madrid, Spain
Ondrej Jakl	Institute of Geonics, Czech Academy of Sciences
Emmanuel Jeannot	Inria, France
Bo Kågström	Umea University, Sweden
Alexey Kalinov	Cadence Design System, Russia
Aneta Karaivanova	Bulgarian Academy of Sciences, Sofia
Eleni Karatza	Aristotle University of Thessaloniki, Greece
Ayse Kiper	Middle East Technical University, Turkey
Jacek Kitowski	Institute of Computer Science, AGH, Poland
Jozef Korbicz	University of Zielona Góra, Poland
Stanislaw Kozielski	Silesia University of Technology, Poland
Dieter Kranzlmueller	Ludwig Maximillian University, Munich, and Leibniz Supercomputing Centre, Germany
Henryk Krawczyk	Gdańsk University of Technology, Poland
Piotr Krzyżanowski	University of Warsaw, Poland
Mirosław Kurkowski	Częstochowa University of Technology, Poland
Krzysztof Kurowski	PSNC, Poznań, Poland
Jan Kwiatkowski	Wrocław University of Technology, Poland
Jakub Kurzak	University of Tennessee, USA
Giulliano Laccetti	University of Naples Federico II, Italy
Marco Lapegna	University of Naples Federico II, Italy
Alexey Lastovetsky	University College Dublin, Ireland
Joao Lourenco	University Nova of Lisbon, Portugal
Hatem Ltaief	KAUST, Saudi Arabia
Emilio Luque	Universitat Autonoma de Barcelona, Spain
Vyacheslav I. Maksimov	Ural Branch, Russian Academy of Sciences
Victor E. Malyshkin	Siberian Branch, Russian Academy of Sciences
Pierre Manneback	University of Mons, Belgium
Tomas Margalef	Universitat Autonoma de Barcelona, Spain
Svetozar Margenov	Bulgarian Academy of Sciences, Sofia
Ami Marowka	Bar-Ilan University, Israel
Norbert Meyer	PSNC, Poznań, Poland
Jarek Nabrzyski	University of Notre Dame, USA
Raymond Namyst	University of Bordeaux and Inria, France
Maya G. Neytcheva	Uppsala University, Sweden
Gabriel Oksa	Slovak Academy of Sciences, Bratislava

Ozcan Ozturk	Bilkent University, Turkey
Tomasz Olas	Częstochowa University of Technology, Poland
Marcin Paprzycki	IBS PAN and SWPS, Warsaw, Poland
Dana Petcu	West University of Timisoara, Romania
Enrique S. Quintana-Ortí	Universidad Jaime I, Spain
Jean-Marc Pierson	Paul Sabatier University, France
Thomas Rauber	University of Bayreuth, Germany
Paul Renaud-Goud	Inria, France
Jacek Rokicki	Warsaw University of Technology, Poland
Gudula Runger	Chemnitz University of Technology, Germany
Leszek Rutkowski	Częstochowa University of Technology, Poland
Robert Schaefer	Institute of Computer Science, AGH, Poland
Olaf Schenk	Università della Svizzera Italiana, Switzerland
Stanislav Sedukhin	University of Aizu, Japan
Franciszek Seredyński	Cardinal Stefan Wyszyński University in Warsaw, Poland
Happy Sithole	Centre for High Performance Computing, South Africa
Jurij Silc	Jozef Stefan Institute, Slovenia
Karolj Skala	Ruder Boskovic Institute, Croatia
Peter M.A. Sloot	University of Amsterdam, The Netherlands
Leonel Sousa	Technical University of Lisbon, Portugal
Radek Stompor	Université Paris Diderot and CNRS, France
Przemysław Stpiczyński	Maria Curie Skłodowska University, Poland
Maciej Stroiński	PSNC, Poznań, Poland
Ireneusz Szcześniak	Częstochowa University of Technology, Poland
Boleslaw Szymanski	Rensselaer Polytechnic Institute, USA
Domenico Talia	University of Calabria, Italy
Christian Terboven	RWTH Aachen, Germany
Andrei Tchernykh	CICESE Research Center, Ensenada, Mexico
Suleyman Tosun	Ankara University, Turkey
Roman Trobec	Jozef Stefan Institute, Slovenia
Denis Trystram	Grenoble Institute of Technology, France
Marek Tudruj	Polish Academy of Sciences and Polish-Japanese Institute of Information Technology, Warsaw, Poland
Bora Uçar	Ecole Normale Superieure de Lyon, France
Marian Vajtersic	Salzburg University, Austria
Jerzy Waśniewski	Technical University of Denmark
Bogdan Wiszniewski	Gdańsk University of Technology, Poland
Andrzej Wyszogrodzki	IMGW, Warsaw, Poland
Ramin Yahyapour	University of Göttingen/GWDG, Germany
Jianping Zhu	Cleveland State University, USA
Julius Žilinskas	Vilnius University, Lithuania
Jarosław Żola	Rutgers University, USA

Contents – Part II

Contents – Part I

Workshop on Scheduling for Parallel Computing (SPC 2013)

Scheduling Bag-of-Tasks Applications to Optimize Computation Time and Cost

Anastasia Grekioti and Natalia V. Shakhlevich(✉)

School of Computing, University of Leeds, Leeds LS2 9JT, UK
{scag,n.shakhlevich}@leeds.ac.uk

Abstract. Bag-of-tasks applications consist of independent tasks that can be performed in parallel. Although such problems are well known in classical scheduling theory, the distinctive feature of Grid and cloud applications is the importance of the cost factor: in addition to the traditional scheduling criterion of minimizing computation time, in Grids and clouds it also important to minimize the cost of using resources. We study the structural properties of the time/cost model and explore how the existing scheduling techniques can be extended to handle the additional cost criterion. Due to the dynamic nature of distributed systems, one of the major requirements to scheduling algorithms is related to their speed. The heuristics we propose are fast and, as we show in our experiments, they compare favourably with the existing scheduling algorithms for distributed systems.

Keywords: Scheduling · Bag-of-tasks applications · Heuristics

1 Introduction

An important feature of the Grid and cloud infrastructure is the need to deliver the required Quality of Service to its users. Quality of Service is the ability of an application to have some level of assurance that users' requirements can be satisfied. It can be viewed as an agreement between a user and resource provider to perform an application within a guaranteed time frame at a pre-specified cost. As a rule, the higher the cost paid by the user, the faster resources can be allocated and the smaller execution time a resource provider can ensure.

While time optimization is the traditional criterion in Scheduling Theory, the cost optimization is a new criterion arising in Grid and cloud applications. In fact the cost- and time-factors are in the centre of resource provision on a pay-as-you-go basis.

The problem we study arises in the context of Bag-of-Tasks applications. It can be seen as the problem of scheduling a set of independent tasks on multiple processors optimizing computation time and cost. Formally, a BoT application consists of n tasks $\{1, 2, \ldots, n\}$ with given processing requirements $\tau_j, j = 1, \ldots, n$, which have to be executed on m uniform processors $P_k, k = 1, \ldots, m$, with speed- and cost-characteristics s_k and c_k. For a BoT application a deadline D is given, so that all tasks of the application should be completed by time D.

Each processor may handle only one task at a time. If task j is assigned to processor P_k, then it should be completed without preemption; the time required for its completion is τ_j/s_k and the cost of using processor P_k is $c_k \times \tau_j/s_k$, or equivalently $\bar{c}_k\tau_j$, where c_k is the cost of using processor P_k per second, while $\bar{c}_k = c_k/s_k$ is the cost of using processor P_k per million instructions. To distinguish between the two types of cost-parameters, we call the initial given cost-values c_k as *absolute costs* and the adjusted values $\bar{c}_k = c_k/s_k$ as *relative costs.*

If $0-1$-variables x_{kj} represent allocation of tasks $j, 1 \leq j \leq n$, to processors $P_k, k = 1, \ldots, m$, then the makespan of the schedule $C_{\max}$ and its cost K are found as

$$C_{\max} = \max_{k=1,\ldots,m} \left\{ \sum_{j=1}^{n} \frac{\tau_j}{s_k} \times x_{kj} \right\}, \qquad K = \sum_{k=1}^{m} \bar{c}_k \sum_{j=1}^{n} \tau_j x_{kj}.$$

The problem consists in allocating tasks to processors so that the execution of the whole application is completed before the deadline D and the cost K is minimum. Adopting standard scheduling three-field notation, we denote the problem by $Q|C_{\max} \leq D|K$, where Q in the first field classifies the processors as uniform, having different speeds, $C_{\max} \leq D$ in the second field specifies that the BoT application should be completed by time D, and K in the third field specifies the cost minimization criterion. Note that developing successful cost-optimization algorithms for problem $Q|C_{\max} \leq D|K$ is useful for solving the bicriteria version of our problem, $Q||(C_{\max}, K)$. Such an algorithm, if applied repeatedly to a series of deadline-constrained problems with different values of deadline D, produces a time/cost trade-off which can be used by a decision maker to optimize two objectives, computation time $C_{\max}$ and computation cost K.

In this study, we explore the structural properties of the time/cost model. Based on these properties, we identify the most appropriate scheduling techniques, developed for makespan minimization, and adopt them for cost optimization. In our computational experiments we show that our algorithms compare favourably with those known in the area of distributed computing that take into account processors' costs, namely [1,2,7,11].

2 Heuristics

Scheduling algorithms for distributed systems are often developed without recognizing the link with related results from scheduling theory. In particular, the existing time/cost optimization algorithms do not make use of such scheduling approaches as the *Longest Processing Time* (LPT) rule and the *First Fit Decreasing* (FFD) strategy. Both approaches have been a subject of intensive research in scheduling literature and they are recognized as the most successful approximation algorithms for solving problem $Q|C_{\max} \leq D|-$, the version of our problem which does not take into account processor costs, see, e.g., [3,4,8].

Both algorithms, LPT and FFD, prioritize the tasks in accordance with their processing requirements giving preference to longest tasks first. In what follows we assume that the tasks are numbered in the LPT-order, i.e., in the non-increasing order of their processing requirements:

$$\tau_1 \geq \tau_2 \geq \cdots \geq \tau_n. \tag{1}$$

We adopt the LPT- and FFD-strategy for a selected subset of processors, taking into account the deadline D.

Algorithm 'Deadline Constrained LPT'

1. Select unscheduled tasks one by one from the LPT-list (1).
2. Assign a current task to the processor selected from the given subset of processors in such a way so that the task has the earliest completion time.
3. If allocation of a task violates the deadline D, keep the task unscheduled.

The FFD algorithm, initially formulated in the context of bin-packing, operates with a fixed ordering of processors (usually, the order of their numbering) and with a list of tasks in the LPT-order (1). It differs from the 'Deadline Constrained LPT' by Step 2, which for FFD is of the form:

2. Assign a current task to the first processor on the list for which deadline D is not violated.

The above two rules are not tailored to handle processor costs. In order to adjust them for solving the cost minimization problem $Q|C_{\max} \leq D|K$, we exploit the properties of the divisible load version of that problem, in which preemption is allowed and any task can be performed by several processors simultaneously. As shown in [9], there exists an optimal solution to the divisible load problem in which processors are prioritized in accordance with their relative costs. If processors are renumbered so that

$$\bar{c}_1 \leq \bar{c}_2 \leq \ldots \leq \bar{c}_m, \tag{2}$$

then in an optimal solution to the divisible load problem, processors $\{P_1, P_2, \ldots, P_{u-1}\}$ with the smallest relative cost $\bar{c}_i$ are fully occupied until time D, one processor P_u with a higher cost can be partly occupied while the remaining most expensive processors $\{P_{u+1}, \ldots, P_m\}$ are free. We call such a schedule as a *box-schedule* emphasizing equal load of the cheapest processors.

In order to solve the non-preemptive version of the problem $Q|C_{\max} \leq D|K$, which corresponds to scheduling a BoT application, we aim to replicate the shape of the optimal box-schedule as close as possible achieving the maximum load of the cheapest processors in the first place. The required adjustment of the FFD algorithm can easily be achieved if processors are renumbered by (2). We call the resulting algorithm 'FFD Cost' (for 'FFD with Processors Prioritized by Cost').

For adjusting the LPT-strategy, notice that it naturally achieves a balanced processor allocation, which in our case is needed only for a subset of cheapest processors. There are several ways of identifying that subset. The two algorithms presented below are based on two different principles of processor selection: in the first algorithm, we group processors with equal relative cost and apply the LPT-strategy to each group; in the second algorithm, we consider one group of r cheapest processors $\{P_1, \ldots, P_r\}$, loading them as much as possible by the LPT-strategy; the remaining processors $\{P_{r+1}, \ldots, P_m\}$ are loaded one by one with unscheduled jobs. The solution of the smallest cost is selected among those constructed for different values of $r, 1 \leq r \leq m$.

Algorithm <u>'LPT Groups'</u> (LPT with Groups of Equivalent Processors)

1. Form groups $G_1, \ldots, G_g$ of processors putting in one group processors of the same relative cost $\bar{c}_i$. Keep group numbering consistent with processor numbering (2) so that the groups with the smallest indices contain the cheapest processors.
2. Consider the groups one by one in the order of their numbering. For each group G_i, apply 'Deadline Constrained LPT' until no more tasks can be allocated to that group.

Algorithm <u>'LPT One Group'</u> (LPT with One Group of Cheapest Processors)

1. For $r = 1, \ldots, m$,
   (a) Select r cheapest processors and apply to them 'Deadline Constrained LPT'.
   (b) For the unscheduled tasks and remaining processors $\{P_{r+1}, \ldots, P_m\}$ apply 'FFD with Processors Prioritized by Cost'.
2. Select r which delivers a feasible schedule of minimum cost and output that schedule.

Since the LPT and FFD strategies have comparable performance in practice, we introduce the counterpart of the latter algorithm with the FFD strategy applied to r cheapest processors. In addition, we use the result from [8] which suggests that the performance of FFD can be improved if the processors in the group are ordered from slowest to fastest.

Algorithm <u>'FFD One Group'</u> (FFD with One Group of Cheapest Processors) 3

1. For $r = 1, \ldots, m$,
   (a) Select r cheapest processors and re-number them in non-decreasing order of their speeds. For the selected processors, apply classical FFD with processors considered in the order of their numbering.
   (b) For the unscheduled tasks and remaining processors $\{P_{r+1}, \ldots, P_m\}$ apply 'FFD with Processors Prioritized by Cost'.
2. Select r which delivers a feasible schedule of minimum cost and output that schedule.

The above heuristics are designed as a modification of time-optimization algorithms. Below we propose one more heuristic based on the cost-optimization algorithm from [10]. It should be noted that unlike most of scheduling algorithms, designed for cost functions depending on task completion times, the algorithm from [10] addresses the model with processor costs.

The approximation algorithm developed in [10] is aimed at solving a more general problem with unrelated parallel machines. This is achieved by imposing restrictions on allocation of long tasks. Depending on the value of the selected parameter t, a task is allowed to be assigned to a processor only if the associated processing time of the task on that processor does not exceed t. We adopt this strategy in the algorithm presented below.

Algorithm 'FFD Long Tasks Restr.' (FFD with Restrictive Allocation of Long Tasks)

```
1. For each combination of $k \in \{1, \dots, m\}$ and $\ell \in \{1, \dots, n\}$ repeat (a)–(c):
   (a) Define parameter $t = \tau_\ell / s_k$ which is used to classify allocation of task $j$
       to processor $i$ as "small" or "large" depending on whether $\tau_j / s_i \le t$ or
       not.
   (b) Consider processors one by one in the order of their numbering (2). Repeat
       for each processor $P_i$:
           Allocate to $P_i$ as many "small" unscheduled tasks from the LPT-list
           (1) as possible, without violating deadline $D$.
   (c) If all tasks are allocated, calculate the cost $K_t$ of the generated schedule,
       else set $K_t = \infty$.
2. Set $K = \min_{t \in \{\tau_l / s_k\}} \{K_t\}$.
```

To conclude we observe that the proposed heuristics are fast (their time complexity ranges from $O(mn)$ to $O(m^2 n^2)$, assuming $n \ge m$) and therefore are suitable for practical applications. In the next section we evaluate the performance of our heuristics empirically comparing them with the known cost-optimization algorithms.

3 Computational Experiments

We have performed extensive computational experiments in order to evaluate the performance of our algorithms and to compare them with the published ones. Figure 1 represents the situation we have observed in most experiments.

The experiments are based on two data sets for processors, denoted by PrI and PrII, and two types of data sets for BoT applications consisting of n tasks, $n \in \{25, 50\}$. Combining the data sets for processors with those for tasks we generate four classes of instances, denoted by PrI-25, PrI-50 and PrII-25, PrII-50. It should be noticed that we have explored the performance of the algorithms on larger instances as well. While the overall behaviour is similar, the differences in the performance of algorithms become less noticeable. This observation is in agreement with the theoretical result on asymptotic optimality of the algorithms

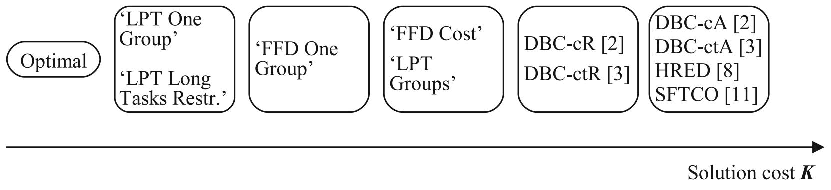


Fig. 1. Algorithm performance

we propose. In a recent study [5] on the workloads of over fifteen grids between 2003 and 2010, it is reported that the average number of tasks per BoT is between 2 and 70 while in most grids the average is between 5 and 20. In order to analyse the distinctive features of the algorithms, in what follows we focus on instances with $n \in \{25, 50\}$.

Data set PrI for processors uses the characteristics of $m = 11$ processors from [1]; data set PrII uses the characteristics of $m = 8$ processors from [6] which correspond to US-West Amazon EC2.

The data sets for tasks are generated in the same fashion for $n = 25$ and $n = 50$. For each value of n, 100 data sets have been created; in each data set the processing requirements of the tasks are random integers sampled from a discrete uniform distribution, as in [1,2], so that $\tau_j \in [1\ 000,\ 10\ 000]$ for $j = 1, \ldots, n$.

The generated processors' data sets and tasks' data set are then combined to generate 4 classes of instances, each class containing 100 tasks' data sets.

Using the data sets for processors and tasks, the instances are produced by considering each combination of data sets and introducing a series of deadlines D for each such combination. The range of deadlines for each combination is defined empirically making sure that for the smallest deadline is feasible (i.e., there exists a feasible schedule for it) and that the largest deadline does not exceed the maximum load of one processor (assuming that all tasks can be allocated to it). Since the deadline ranges differ, the actual number of instances for each combination of processors' and tasks' data sets varies greatly: there 1579 instances in class PrI-25, 3093 instances in PrI-50, 861 instances in PrII-25 and 1645 instances in PrII-50.

The comparison is done with the following algorithms known in the context of Grid scheduling:

- **'DBC-cA'** for the version of the 'Deadline and Budget Constrained (DBC) cost optimization' algorithm by Buyya & Murshed [1], which uses processors' absolute costs [1],
- **'DBC-cR'** for the version of the previous algorithm, which uses processors' relative costs,
- **'DBC-ctA'** for the version of the 'Deadline and Budget Constrained (DBC) cost-time optimization' algorithm by Buyya et al. [2], which uses processors' absolute costs,
- **'DBC-ctR'** for the version of the previous algorithm, which uses processors' relative costs,

– **‘HRED’** for the algorithm ‘Highest Rank Earliest Deadline’ by Kumar et al. [7],
– **‘SFTCO’** for the algorithm ‘Stage Focused Time-Cost Optimization’ by Sonmez & Gursoy [11].

We have implemented all algorithms in Free Pascal 2.6.2 and tested them on a computer with an Intel Core 2 Duo E8400 processor running at 3.00 GHz and 6 GB of memory. All heuristics are sufficiently fast to be applied in practice. Considering the heuristics we propose, the fastest two are **‘LPT Groups’** and **‘FFD Cost’** which solve instances with 50 tasks in about 0.02 s; the actual running time of heuristics **‘LPT One Group’** and **‘FFD One Group’** does not exceed 0.41 s for all instances, while **‘FFD Long Tasks Restr.’** has the highest time complexity and runs on large instances for 4.28 s. As far as the existing algorithms are concerned, the five heuristics **‘DBC-cA’**, **‘DBC-ctA’**, **‘DBC-cR’**, **‘DBC-ctR’** and **‘SFTCO’** are comparable with our two fastest heuristics producing results in 0.01–0.02 s; the running time of **‘HRED’** is 1–1.45 s.

The summary of the results is presented in Tables 1, 2 and 3. In each table, the best results for each class of instances are highlighted in bold. In Table 1 we present the percentage of instances for which each algorithm produces no-worse solutions (of a lower or equal cost) and the percentage of instances for which each algorithm produces strictly better solutions (of a strictly lower cost) in comparison with any other algorithm; the latter percentage value is given in the parentheses. For each class of instances we present with bold numbers the best performance for the known and new algorithms. Notice that the best results are achieved by algorithms **‘LPT One Group’** and **‘FFD Long Tasks Restr.’**.

While Table 1 indicates that the algorithms we propose are in general of better performance, the actual entries in the table do not reflect to which extent the new algorithms are superior in comparison with the known algorithms.

Table 1. Percentage of instances for which algorithms produce no worse solutions and strictly better solutions, in parentheses

Grid scheduling algorithms	PrI-25	PrI-50	PrII-25	PrII-50
DBC-ctA [2]	2.44(0)	1.6(0)	0(0)	0(0)
DBC-ctR [2]	2.44(0)	1.6(0)	6.86(0)	5.2(0)
DBC-cA [1]	**4.72(0)**	**3.74(0)**	0(0)	0(0)
DBC-cR [1]	**4.72(0)**	**3.74(0)**	**7.32(0)**	**5.78(0.06)**
HRED [7]	1.29(0.07)	0.71(0.05)	6.71(0)	4.45(0)
SFTCO [11]	0(0)	0(0)	0(0)	0(0)
Proposed algorithms	PrI-25	PrI-50	PrII-25	PrII-50
FFD One Group	18.9(5.41)	18.16(4.62)	12.73(2.36)	9.99(0.92)
LPT One Group	**23.01(2.88)**	**26.25(4)**	14.63(2.29)	13.75(2.37)
LPT Groups	11.99(1.06)	12.42(0.85)	9.68(0)	8.26(0.06)
FFD Cost	13.49(0)	13.54(0)	10.37(0)	9.07(0)
FFD Long Tasks Restr.	17.01(3.5)	18.25(4.71)	**31.71(19.59)**	**43.5(33.68)**

A general comparison of the two groups of algorithms, the known and the new ones, is performed in Table 2. In the first column we list 4 classes of instances considered in the experiments. In the second column we provide the percentage of instances in each class, for which the best solution (in terms of the strictly lower cost) is found by one of the existing algorithms. In the third column we provide the percentage of instances in each class, for which the best solution is found by one of the new algorithms. The last column specifies the percentage of instances for which the best solutions found by the existing algorithms and the new ones have the same cost. It is easy to see that the algorithms we propose outperform the existing ones producing strictly lower cost schedules in more than 85 % of instances.

A special attention should be given to instances with tight deadlines. Our experiments show that the new algorithms fail to produce feasible schedules meeting a given deadline in less than 0.46 % of instances in all classes, while for the existing algorithms the number of fails range between 0.43 %–100 %.

Our next table (Table 3) evaluates the quality of the solutions produced by various algorithms. It is based on the values of the relative deviation

$$\theta = 100 \times (K - LB)/LB,$$

in percentage terms, of cost K of heuristic solutions from the lower bound values LB. For each instance, the LB-value is found as the cost of the box-schedule optimal for the relaxed model with preemption allowed.

The maximum and the average deviation is then calculated over all instances in each class reflecting the worst and the average performance of each algorithm. The best results for each class of instances are highlighted in bold for the known and new algorithms. It should be noted that the entries in the last two columns for algorithm SFTCO are empty as that algorithm failed to produce deadline feasible solutions in all instances of classes PrII-25 and PrII-50.

The algorithms we propose have average percentage deviation from the lower bound between 0.03 %–1.09 %, while the same figures for the known algorithms are between 0.58 %–162 %. The worst case performance of algorithms, measured by the maximum percentage deviation from the lower bound, is between 0.31 %–5.74 % for our algorithms and between 4.16 %–287.4 % for known ones. Thus in all scenarios our algorithms outperform the known Grid scheduling heuristics.

Table 2. Percentage of strictly lower cost solutions and ties for existing and proposed algorithms

	Lowest cost (existing)	Lowest cost (proposed)	Ties
PrI-25	2.91	85.12	11.97
PrI-50	4.75	88.78	6.47
PrII-25	9.76	88.50	1.74
PrII-50	4.98	94.41	0.61

Table 3. The maximum relative deviation θ (in percentage) of cost K of heuristic solutions from LB (the average relative deviation in parentheses)

Grid scheduling algorithms	PrI-25	PrI-50	PrII-25	PrII-50
DBC-ctA [2]	17.55(4.33)	7.27(1.56)	81.56(39.47)	79.92(35.28)
DBC-ctR [2]	17.55(4.40)	7.27(1.65)	9.81(2.16)	4.90(0.86)
DBC-cA [1]	**13.80(2.56)**	**6.44(0.95)**	68.51(29.26)	66.56(24.99)
DBC-cR [1]	14.58(2.68)	6.44(1.11)	**9.57(1.4)**	**4.16(0.58)**
HRED [7]	19.77(9.19)	9.49(4.41)	15.58(5.43)	7.19(2.59)
SFTCO [11]	287.4(162)	275.4(137)	-(-)[a]	-(-)[a]
Proposed algorithms	PrI-25	PrI-50	PrII-25	PrII-50
FFD One Group	5.23(0.75)	1.04(0.25)	2.11(0.41)	0.99(0.14)
LPT One Group	4.32(**0.73**)	**1.04(0.21)**	1.79(0.29)	0.99(0.09)
LPT Groups	4.99(1.09)	2.22(0.48)	2.9(0.51)	0.99(0.19)
FFD Cost	5.74(0.89)	1.29(0.31)	2.11(0.46)	0.99(0.17)
FFD Long Tasks Restr.	**3.6**(0.78)	1.04(0.26)	**1.35(0.13)**	**0.31(0.03)**

[a] No feasible solutions found by Algorithm SFTCO in all instances of classes PrII-25 and PrII-50.

One more observation can be done in relation to the asymptotic optimality of our algorithms. Table 3 supports the fact that as the number of tasks increases from 25 to 50, the solutions found become closer to lower bounds. The proof of the asymptotic optimality of the algorithms is omitted in the current paper.

The performance of the known algorithms can be summarized as follows. The best amongst those algorithms are the **DBC**-algorithms from [1] and [2]. Algorithms **DBC-cA** and **DBC-ctA** load the processors giving preference to those with the lowest absolute cost c_k, while **DBC-cR** and **DBC-ctR** give preference to the processors with the lowest relative cost $\bar{c}_k$, approximating the box-schedule introduced in Sect. 2. Due to the optimality of the latter schedule for the preemptive version of the problem [9], it is expected that **DBC-cR** and **DBC-ctR** should outperform **DBC-cA** and **DBC-ctA**. That behaviour is clearly observed in our experiments with processors' data set PrII; experiments with data set PrI are less indicative since in PRI the orderings of processor based on c_k and $\bar{c}_k$ are very similar.

The algorithms from [1] and [2] consider the tasks without taking into account their processing requirements, while all our algorithms make use of the LPT task order. The algorithm from [7] performs task allocation in a fashion similar to the SPT order rather than LPT, which might explain its poor performance. Finally, the algorithm from [11] prioritizes the tasks in the LPT order, but the authors use a strange ratio $s_i/\bar{c}_i = s_i^2/c_i$ to prioritize the processors; the meaning of that ratio and its justification are not provided.

Analysing the performance of our algorithms we conclude that the best performance is achieved by either **LPT One Group** or **FFD Long Tasks Restr.**. In the instances with tight deadlines, algorithm **LPT One Group** outperforms the others as it is more successful in finding deadline feasible schedules. The third best algorithm is **FFD One Group**. Finally, a slightly worse performance of algorithms **LPT Groups** and **FFD Cost** is compensated by their faster running times.

4 Conclusions

In this study we propose new cost optimization algorithms for scheduling Bag-of-Tasks applications and analyse them empirically. Our algorithms are aimed at replicating the shape of a box-schedule, which is optimal for a relaxed version of the problem. This is achieved by combining the most successful classical scheduling algorithms and adjusting them for handling the cost factor.

In comparison with the existing algorithms, new algorithms find strictly lower cost schedules in more than 85 % of the instances; they are fast and easy to implement and can be embodied in a Grid broker.

Acknowledgements. This research was supported by the EPSRC funded project EP/G054304/1 "Quality of Service Provision for Grid applications viaIntelligent Scheduling".

References

1. Buyya, R., Murshed, M.: GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurr. Comput. Pract. Exp. **14**, 1175–1220 (2002)
2. Buyya, R., Murshed, M., Abramson, D., Venugopal, S.: Scheduling parameter sweep applications on global grids: a deadline and budget constrained cost-time optimization algorithm. Softw. Pract. Exp. **35**, 491–512 (2005)
3. Coffman, E.G., Garey, M.R., Johnson, D.S.: Application of Bin-Packing to Multiprocessor Scheduling. SIAM J. Comput. **7**, 1–17 (1978)
4. Gonzalez, T., Ibarra, O.H., Sahni, S.: Bounds for LPT schedules on uniform processors. SIAM J. Comput. **6**, 155–166 (1977)
5. Iosup, A., Epema, D.: Grid computing workloads. IEEE Internet Comput. **15**, 19–26 (2011)
6. Javadi, B., Thulasiram, R., Buyya, R.: Statistical modeling of spot instance prices in public cloud environments. In: 4th IEEE International Conference on Utility and Cloud Computing, Melbourne, Australia, pp. 219–228 (2011)
7. Kumar, S., Dutta, K., Mookerjee, V.: Maximizing business value by optimal assignment of jobs to resources in grid computing. Eur. J. Oper. Res. **194**, 856–872 (2009)
8. Kunde, M., Steppat, H.: First fit decreasing scheduling on uniform multiprocessors. Discrete Appl. Math. **10**, 165–177 (1985)
9. Shakhlevich, N.V., Djemame, K.: Mathematical models for time/cost optimization in grid scheduling. Report 2008.04, School of Computing, University of Leeds (2008). http://www.engineering.leeds.ac.uk/computing/research/publications/reports/2008+/2008_04.pdf
10. Shmoys, D.B., Tardos, É.: An approximation algorithm for the generalized assignment problem. Math. Program. **62**, 461–474 (1993)
11. Sonmez, O.O., Gursoy, A.: A novel economic-based scheduling heuristic for computational grids. Int. J. High Perform. Comput. Appl. **21**, 21–29 (2007)

Scheduling Moldable Tasks with Precedence Constraints and Arbitrary Speedup Functions on Multiprocessors

Sascha Hunold(✉)

Research Group Parallel Computing,
Vienna University of Technology, Vienna, Austria
hunold@par.tuwien.ac.at

Abstract. Due to the increasing number of cores of current parallel machines, the question arises to which cores parallel tasks should be mapped. Thus, parallel task scheduling is now more relevant than ever, especially under the moldable task model, in which tasks are allocated a fixed number of processors before execution. Scheduling algorithms commonly assume that the speedup function of moldable tasks is either non-decreasing, sub-linear or concave. In practice, however, the resulting speedup of parallel programs on current hardware with deep memory hierarchies is most often neither non-decreasing nor concave.

We present a new algorithm for the problem of scheduling moldable tasks with precedence constraints for the makespan objective and for arbitrary speedup functions. We show through simulation that the algorithm not only creates competitive schedules for moldable tasks with arbitrary speedup functions, but also outperforms other published heuristics and approximation algorithms for non-decreasing speedup functions.

Keywords: Multiprocessor scheduling · Homogeneous processors · Moldable tasks · Makespan optimization · Speedup functions

1 Introduction

The problem of scheduling parallel tasks has been intensively studied, and it originally stems from the need of improving the utilization of massively parallel processors (MPPs) [1]. Researchers attempted to understand the implications of different job scheduling strategies on the utilization of parallel systems theoretically and practically. Drozdowski distinguishes between three *types of parallel tasks* [2] (called *job flexibility* by Feitelson et al. [1]): (1) the *rigid task* requires a predefined fixed number of processors for execution, (2) the *moldable* task for which the number of processors is decidable before the execution starts, but stays unchanged afterwards, and (3) the *malleable* task, where the number of processors may vary during execution.

We focus on the *moldable* task model. The reason is mostly practical, since the malleable model would require additional effort from programmers, e.g., to

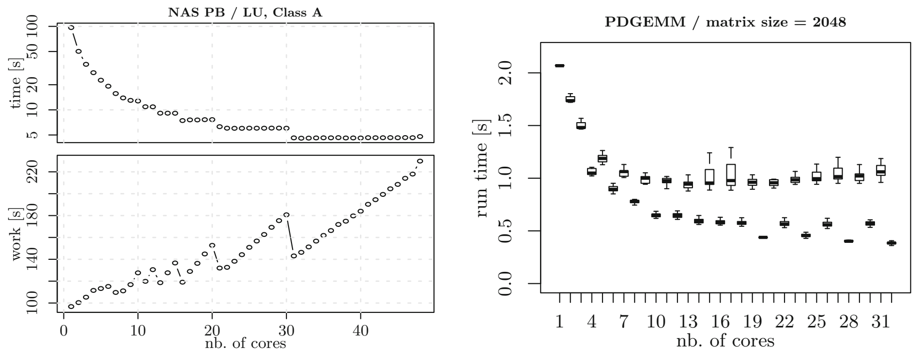


Fig. 1. Left: work (top) and run-time (bottom) of LU benchmark (4 sockets, 48 cores, AMD Opteron 6168); Right: execution times of PDGEMM (20 runs per core count, GBit Ethernet, AMD Opteron 6134)

redistribute data or synchronize thread groups. Pthreads or OpenMP programs are typical examples of moldable tasks as users can specify the number of threads before the execution of a parallel program. MPI applications are examples of moldable programs for distributed memory machines.

Today, researchers in parallel computing face the question of how to program the available processors (or cores) efficiently. One approach is to model an application as *directed cyclic graphs (DAGs)*, where edges make data dependencies explicit and nodes represent computations. The MAGMA library is an example, where DAGs represent parallel applications [3].

To execute moldable tasks of a DAG, a scheduling algorithm has to determine (1) the next task to be executed and (2) the set of processors to which a task is mapped. In the scheduling literature, this is known as *scheduling problem for moldable tasks with precedence constraints* [2] (sometimes also called malleable task scheduling [4–7]). A common assumption is that the run-time function of each parallel task is non-increasing and the corresponding work function is non-decreasing in the number of processors. The work is defined as the product of run-time and number of processors. Figure 1 shows two examples where this assumption is violated in practice. The two plots on the left show run-time and work of the NAS LU benchmark on a 48-core shared-memory machine (median of five runs). The run-time is almost non-increasing but the corresponding work decreases several times, e.g., at 21 or 31 cores. The chart on the right shows the run-time of PDGEMM from ScaLAPACK (using GotoBLAS). Since this matrix multiplication routine works best on a square processor grid, we see an increased run-time for 11 or 13 processors. This "zigzagging" was already observed by van de Geijn and Watts [8]. One could solve this problem of the 0run-time function by using $(k-1)$ instead of k processors if the run-time on $(k-1)$ processors is smaller. The resulting run-time function would be non-increasing, but the work function could still be decreasing (similar to the plot on the left-hand side).

Therefore, we propose an *algorithm for scheduling non-preemptive moldable tasks with precedence constraints* for (1) *non-increasing run-time and non-decreasing work functions* and also (2) *arbitrary run-time and work functions.* In the three-field notation of Graham et al., we investigate $P|any, NdSub, prec|C_{\max}$ and $P|any, prec|C_{\max}$, where P denotes the number of processors, *any* the moldable tasks, *prec* the precedence constraints, and $NdSub$ the nondecreasing sublinear speedup[1]. Our objective is to minimize the makespan $C_{\max}$.

We make several contributions to moldable task scheduling. First, we propose a *new algorithm* (CPA13)[2] that *supports arbitrary run-time functions* of moldable tasks. We show through simulation that our algorithm is competitive in the cases considered. Second, we compare schedules produced by CPA13 with schedules produced by several *approximation algorithms with performance guarantees.* To the best of our knowledge, this is the first experimental study that evaluates both CPA-style algorithms and approximation algorithms through simulation. We show that our *new algorithm* CPA13 produces *shorter schedules* even when *the run-time function of each parallel task is non-increasing.* Previous studies of moldable task scheduling algorithms use the absolute makespan to compare heuristics. However, such analysis provides little evidence of the schedule quality. Hence, as third contribution, we compare algorithms by their *performance ratio*, which is the *ratio of makespan and lower bound.*

Section 2 introduces notation and Sect. 3 discusses related work. We introduce the new scheduling algorithm and show complexity results in Sect. 4, while Sect. 5 presents the simulation results before we conclude in Sect. 6.

2 Definitions and Notation

We consider the problem of scheduling n moldable tasks with precedence constraints on m identical processors under the makespan objective $C_{\max}$. We study the offline and clairvoyant version of the problem, i.e., the entire DAG and the processing times for each node are known to the scheduler. The application is represented as directed acyclic graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ denotes the set of moldable tasks and $E \subseteq V \times V$ represents the set of edges (arcs) between tasks ($e = |E|$). For every task v_j, $p_j(i)$ denotes the execution time of task j on i processors, and $w_j(i) = i \cdot p_j(i)$ denotes its work. Further, variable α_j refers to the number of processors allotted to task j.

The functions $p_j(i)$ and $w_j(i)$ are often assumed to be monotonic [10, Chap. 26], i.e., $p_j(i)$ is non-increasing and $w_j(i)$ non-decreasing in i. Formally, $p_j(i) \geq p_j(k)$, and $w_j(i) \leq w_j(k)$, for $i \leq k$. Some algorithms require that $p_j(i)$ not only needs to be non-increasing, but also *convex* in the interval $[1, m]$. The work W of a DAG is computed as $W = \sum_{v_i} p_i(\alpha_i)\alpha_i$.

[1] For more details on notation see [2,9,10].

[2] Critical Path and Area-based Scheduling (CPA), "13" refers to the present year.

3 Related Work

The problem of scheduling rigid tasks, where precedence constraints are given as a set of chains $P2|size_j, chain|C_{\max}$ is strongly NP-hard for $m \geq 2$ [10]. For the more general problem of scheduling moldable tasks with precedence constraints several approximation algorithms exist. Lepère et al. presented an approximation algorithm with performance guarantee of $3 + \sqrt{5} \approx 5.236$ [4], where the scheduling problem is decomposed into an allotment and a mapping problem. The allotment problem is solved by applying Skutella's method for obtaining an approximate solution to the discrete time-cost trade-off problem [11]. Jansen and Zhang improved the approximation ratio (to ≈ 4.73) by changing the rounding strategy for the fractional solution produced by the linear relaxation of the discrete problem [5]. The algorithms presented in [4–6] require monotony of runtime and work, and the most recent algorithm (with approximation ratio ≈ 3.29) also requires that "the work function of any malleable task is non-decreasing in the number of processors and is convex in the processing time" [6].

Radulescu and van Gemund used similar observations as reported in [4,5] for solving the allotment problem. Thus, the Critical Path and Area-based Scheduling (CPA) algorithm is based on the fact that the average work W/m and the length of the critical path L are lower bounds of C_{max} [12]. CPA starts by allocating a single processor to each task, inspects the tasks on the critical path, and then adds one processor to the task that decreases the average processor utilization (runtime / number of processors) the most. The allocation process repeats until the critical path L is smaller than the average work (W/m). Bansal et al. discovered that CPA should take task parallelism better into account [13]. More precisely, the allocation routine of CPA often produces large processor allotments, with the consequence that tasks—which can potentially be executed in parallel—need to run sequentially. Bansal et al. introduced the Modified CPA (MCPA) algorithm, which considers the depth of tasks in the allocation phase. In particular, no more processors are allotted to a task if already m processors have been allotted to tasks in the same depth. We showed in previous work how low-cost improvements to MCPA, such as relaxing the allotment constraints per precedence level or allowing allocation reductions, can improve the performance significantly [14].

Desprez and Suter attempted to optimize both the makespan and the total work when scheduling a DAG [15]. They proposed the bi-criteria optimization algorithm BiCPA that computes the processor allotment for m different cluster sizes $[1, 2, \ldots, m]$ and selects the allotment that optimizes a given makespan-work ratio. BiCPA produces "short" and "narrow" schedules, yet it increases the computational complexity significantly.

All algorithms described above assume non-increasing run-time and non-decreasing work functions. For the case of arbitrary run-time functions, only few algorithms have been proposed. Günther et al. presented an FPTAS for this scheduling problem [7]. As the general problem is strongly NP-hard, they looked at DAGs with bounded width and developed a dynamic program. For practical applicability, the FPTAS has a rather large complexity of $O((\frac{n^3}{\epsilon})^{2\omega} m^{2\omega})$, where

Table 1. Overview of notation used in the present article

m	– number of processors	n	– number of tasks (nodes)
e	– number of edges ($\|E\|$)	L	– length of critical path
W	– overall work of DAG	G	– min.rel. run-time improvement
v_j	– task j	α_j	– processors allotted to task v_j
$p_j(k)$	– run-time of task j with k processors	b_j	– benefit of task j
r_j	– rel. run-time improvement	l_j^b	– bottom level of task v_j
l_j^p	– precedence level of task v_j	$\tilde{m}_d$	– nb. processors in prec. level d

ω denotes the maximum width of a DAG. In previous work, we already addressed the challenge of arbitrary run-time functions by using an evolutionary algorithm (EA) to find short schedules [16]. The proposed algorithm EMTS takes allotment solutions of several heuristics, like CPA and MCPA, and optimizes them using an $(\mu + \lambda)$-EA. When generating and evaluating many offspring, EMTS can find short schedules, while having the disadvantage of an increased run-time.

In sum, efficient heuristics and approximation algorithms only exist for non-increasing run-time and non-decreasing work functions, and previous algorithms without such limitations have high computational demands.

4 Scheduling Algorithm

Our proposed scheduling algorithm applies concepts of the algorithms discussed before, e.g., reducing the critical path while keeping the overall work small. Lepère et al. and Jansen/Zhang also applied this concept in their algorithms. We define the following requirements for our scheduling algorithm: several published articles addressed the problem of CPA that allotments can grow too big regardless of their speedup. To solve this problem, we introduce the *relative run-time threshold G*, which defines the minimum runtime improvement that a larger allotment needs to provide to be considered as possible solution. As shown later, this threshold is key for short and compact schedules. Task parallelism should be conserved as much as possible. To do so, MCPA checks all the visited nodes in a certain DAG depth, but may unmark once visited nodes. In contrast, our algorithm considers all allotments of those nodes that were once marked. In addition, the mapping function that selects processors for ready tasks has often been overlooked in previous studies. Since we showed that reducing processor allotments in the mapping step can significantly improve the overall schedule [14], CPA13 applies a binary search strategy to find a possibly smaller task allotment that does not increase the estimated completion time.

4.1 Pseudocode

Algorithm 1 presents the allotment function of our algorithm named CPA13. For an easier comprehension we summarize the notation in Table 1. Let us highlight the main steps of the algorithm. In the initialization phase, each task is allotted

Algorithm 1. CPA13 allocation procedure

```
1: for all $v_j \in V$ do
2:     $\alpha_j \leftarrow 1$
3:     mark $v_j$ as UNVISITED
4:     $A_j \leftarrow$ list of increasing allotment sizes for which:
               $\forall a_i, a_k \in A_j, i < k : p_j(i) > p_j(k)$
5:     $\tilde{k} \leftarrow 1$
6:     for $k$ in $2 \ldots |A_j|$ do
7:         $b_{j_k} \leftarrow \left( \frac{p_j(a_{\tilde{k}})}{a_{\tilde{k}}} - \frac{p_j(a_k)}{a_k} \right)$
8:         $r_{j_k} \leftarrow \frac{p_j(a_{\tilde{k}}) - p_j(a_k)}{p_j(a_{\tilde{k}})}$
9:         if $r_{j_k} \geq G$ then
10:            $\tilde{k} \leftarrow k$
11:            store $(k, b_{j_k})$ in list of possible allotments for task $v_j$
12: while $L > W/m$ do
13:     $V_{CP} \leftarrow$ collect tasks on critical path
14:     $(v_b, \tilde{\alpha_b}, b_b) \leftarrow (\mathbf{nil}, m, 0.0)$            // initialize current best temporary values
15:     $\tilde{m}_d \leftarrow \sum_{v_l \in \tilde{V}} \alpha_l$ where $\tilde{V} = \{v_k \in V$ s.t. $l_k^b = d \wedge v_k$ marked VISITED $\}$
16:     for all $v_j \in V_{CP}$ do
17:         $b_{j_k}, a_{j_k} \leftarrow$ benefit and size of task's $v_j$ next larger allocation
18:         $s \leftarrow a_{j_k} - \alpha_j$                // absolute increase in number of processors
19:         if $\tilde{m}_{d_j} + s \leq m \wedge b_{j_k} > b_b$ then
20:            $(v_b, \tilde{\alpha_b}, b_b) \leftarrow (v_j, a_{j_k}, b_{j_k})$                          // current best
21:     if $v_b \neq \mathbf{nil}$ then
22:         $\alpha_b \leftarrow \tilde{\alpha_b}$                              // increase allotment of task $v_b$
23:         mark $v_b$ as VISITED
24:         recompute $L$ and $W$
25:     else
26:         break                                         // terminate while loop
```

one processor and marked unvisited. We also pre-compute the possible benefit and the relative execution time reduction of each processor allotment (line 6–11).

In the second phase (line 12), we compute the critical path and select the task with the greatest benefit value. We allot more processors to this task unless the number of processes in this precedence level is exceeded (lines 19–20) (The precedence level denotes the shortest path to a node from the source node). After changing the allotment of one task on the critical path, we need to recompute L and W. The allotment process repeats until either the critical path L is smaller than (or equal to) W/m or no more task satisfies the precedence level constraint or provides additional run-time benefit.

Algorithm 2 presents the mapping procedure of CPA13, which first considers all ready tasks and extracts the task with highest priority. We use the highest bottom level as priority, i.e., the longest path from a node to the sink of the DAG. After extracting the ready task v_j, the procedure selects the α_j processors that first become idle. However, this allotment of v_j might be packed (decreased in size) without increasing its completion time. In order to find such a smaller processor allotment for v_j we perform a binary search on v_j's allotments.

Algorithm 2. CPA13 mapping procedure

```
1: while not all tasks scheduled do
2:     $v_j \leftarrow$ find tasks with maximum $l_j^b$
3:     LET $C_j = \tau_j + p_j(\alpha_j)$ be the completion time of $v_j$
                  if $v_j$ is allocated $\alpha_j$ processors that become available first at time $\tau_j$
4:     $\alpha_i \leftarrow$ use binary search to find an allocation $\alpha_i \leq \alpha_j$ s.t. $\tau_i + p_j(\alpha_i) \leq C_j$
5:     schedule $v_j$ onto first $\alpha_i$ processors that become available
```

Table 2. Summary of complexity results, "t.p." stands for "this paper"

Algorithm	Allocation procedure		Mapping procedure	
CPA	[12]	$O(nm(n+e))$	[12]	$O(n \log n + nm + e)$
MCPA	[13]	$O(nm(n+e))$	[12]	$O(n \log n + nm + e)$
BiCPA-S	[15]	$O(nm(n+e))$	[15]	$O(m(n \log n + nm + e))$
JZ06	[5]	$O(LP(mn, n^2 + mn))$	[4]	$O(mn + e)$
JZ12	[6]	$O(LP(mn, n^2 + mn))$	[4]	$O(mn + e)$
CPA+NM+R	t.p.	$O(nm(n+e))$	[12]	$O(n \log n + nm + e)$
MCPA+NM+R	t.p.	$O(nm(n+e))$	[12]	$O(n \log n + nm + e)$
EMTS	[16]	input dependent	[12]	$O(r(n \log n + nm + e))$
CPA13	t.p.	$O(nm(n+e))$	[14]	$O(n(\log n + m \log m) + e)$

4.2 Asymptotic Run-Time Analysis

We determine the run-time complexity of CPA13 (the number of operations to perform) by examining the allocation and the mapping step separately. In the allocation phase of CPA13 (Algorithm 1), the benefit of a processor allotment is computed for all tasks ($O(nm)$). The body of the loop (line 12) determines the number of processors per precedence level ($O(n)$) and the critical path ($O(n+e)$). After selecting and modifying the best task, the critical path needs to be updated (($O(n + e)$). The outer loop (line 12) is executed at most $n \cdot m$ times since then each tasks will have m processors allotted to it. Thus, the complexity of the allocation phase is $O(nm(n + e))$.

The mapping procedure (Algorithm 2) first extracts the task with highest priority ($O(\log n)$ using a heap) and selects the processors that become idle next ($O(m)$). We apply a binary search ($O(\log m)$) on the processors, but which need to be sorted by increasing finishing time first ($O(m \log m)$). Upon mapping a task, the algorithm visits every outgoing edge to detect ready tasks. In total over all iterations, $O(e)$ edges are visited in the procedure. Given that the loop in line 1 runs once for every task, the overall complexity of the mapping procedure is $O(n(\log n + m \log m) + e)$.

Additionally, we present the asymptotic run-times of both the allocation and mapping procedure of related algorithms in Table 2. JZ06 and JZ12 denote the algorithms of Jansen and Zhang from 2006 [5] and 2012 [6]. The authors state that the LIST scheduling function requires $O(nm)$ operations, while $LP(p, q)$ denotes "the time to solve a linear program with p variables and q constraints" [5].

As the number of edges may be greater than mn, we updated this run-time to $O(mn+e)$. The suffix “NM+R” behind CPA and MCPA identifies our modified versions, which are aware of possibly increasing run-time functions (discussed in Sect. 5). The evolutionary algorithm EMTS is input-dependent as it takes solutions of other heuristics for obtaining the initial population, and its run-time grows with the number of generations produced in the optimization process. Thus, EMTS calls the mapping function for each individual, and r denotes the total number of individuals created.

5 Evaluation

We use simulation to evaluate CPA13 for two reasons: (1) Simulations allows us to obtain a statistically significant number of results. (2) Not many truly moldable applications exist, which would limit the variety of experiments.

5.1 DAGs and Platforms

We consider two types of DAGs in the simulation: (1) application DAGs that mimic existing parallel algorithms and (2) synthetic random DAGs. Strassen’s matrix multiplication algorithm and the Fast Fourier Transformation (FFT) are examples of *application-oriented DAGs*. To obtain different computation and scalability ratios, we keep the shape of these DAGs fixed but change the number of operations of each task. The *synthetic DAGs* are generated with DAGGEN [17] and contain 20, 50 or 100 nodes. Four parameters influence the DAG generation process: the *width* controls how many task can run in parallel, the *regularity* defines the uniformity of the number of tasks per DAG level, *density* specifies the number of edges, and *jump* denotes if and how many DAG levels an edge (arc) may span. In total, we created 400 FFT, 100 Strassen, 108 layered and 324 irregular DAGs. *Layered DAGs* have edges only between adjacent precedence levels ($jump = 0$) and the tasks in one tree level have an equal number of operations.

The number of operations per task depends on a data size d (number of elements) and a function applied to the data, which were both randomly selected. The function $f(d)$ that is applied to the d elements defines the number of operations and is one of the following: stencil – d, sorting – $d \log d$, matrix multiplication – $(\sqrt{d})^3$. Function $f(d)$ and data size d only define the sequential time of a task. To obtain the parallel run-time, we apply Amdahl’s model and pick the non-parallelizable fraction β (see next section) of $f(d)$, which is selected randomly from a uniform distribution between 0 and 0.25. Due to the page limit we refer to [15,16] for more details.

The platform model has two parameters: (1) the number of processors m and (2) the speed of the processor (in GLFOPS). We use two machine models in the simulations. The first models a Grid’5000 cluster (Grelon) with $m = 120$ processors providing 3.1 GFLOPS each (obtained with HP-LinPACK). The other machine has $m = 48$ processors running at 6.7 GFLOPS (measured with GotoBLAS2), modeling a shared-memory system at TU Vienna.

We apply two different run-time models to parallel tasks in our simulation.

Run-time Model 1: Since each task in the DAG generation process is assigned (1) a number of operations to perform and (2) the fraction of non-parallelizable code, we apply Amdahl's law to obtain a run-time model. Let $p_i(1)$ be the sequential run-time of task v_i, determined by the ratio of the number of operations and the speed of the processor. Let β, $0 \leq \beta \leq 1$, be the non-parallelizable fraction of a parallel task, then the run-time of task v_i on k processors is bounded by $p_i^1(k) = (\beta + \frac{1-\beta}{k}) \cdot p_i(1)$. Applying this formula yields a non-increasing run-time and non-decreasing work function for each parallel task. In addition, the run-time function is also convex over the interval $[1, m]$, which is required to apply algorithm JZ12.

Run-time Model 2: Figure 1 has shown that the run-time of PDGEMM is larger with an odd number of processors or if the number of processors has no integer square root. We model the second runtime function accordingly, but base it on the Amdahl model $p_i^1(k)$. Equation (1) shows the runtime function $p_i^2(k)$, which may increase if allocations get larger.

$$p_i^2(k) = \begin{cases} p_i^1(1) & \text{if } k = 1, \\ s_1 \cdot p_i^1(k) & \text{if } k > 1 \wedge k \text{ is odd}, \\ s_2 \cdot p_i^1(k) & \text{if } k > 1 \wedge k \text{ is even, but } \sqrt{k} \text{ not an integer}, \\ p_i^1(k) & \text{otherwise.} \end{cases} \tag{1}$$

s_1 and s_2 are the slowdown factors applied when the number of processors is odd or has no integer square root. In the simulations, we set $s_1 = 1.3$ and $s_2 = 1.1$ to reflect the observed run-time behavior of PDGEMM.

5.2 Simulation Results

Run-time Model 1. The first set of simulations compares algorithms that were designed for non-increasing run-time functions with CPA13. This experiment answers two questions: (1) What is the overall schedule quality of CPA13 compared with the lower bound? (2) How good are CPA13's solutions compared with solutions of approximation algorithms?

In previous studies of CPA, algorithms have been compared without a baseline, so it was uncertain whether experimental findings have significant impact. Since the problem is strongly NP-hard, we use the lower bound as an approximation of the optimum as done by Albers and Schröder [18]. The length of the critical path and the average work per processor are lower bounds of the makespan. Thus, the lower bound of the makespan is $LB = \max\left\{\sum_{j=1}^{n} w_j(1)/m, L^*\right\}$, where L^* denotes the shortest possible critical path. To compute L^* we allocate k processors to task v_j with $k = \arg\min_{l} p_j(l)$, then we compute the critical path using this processor allotment and determine its length.

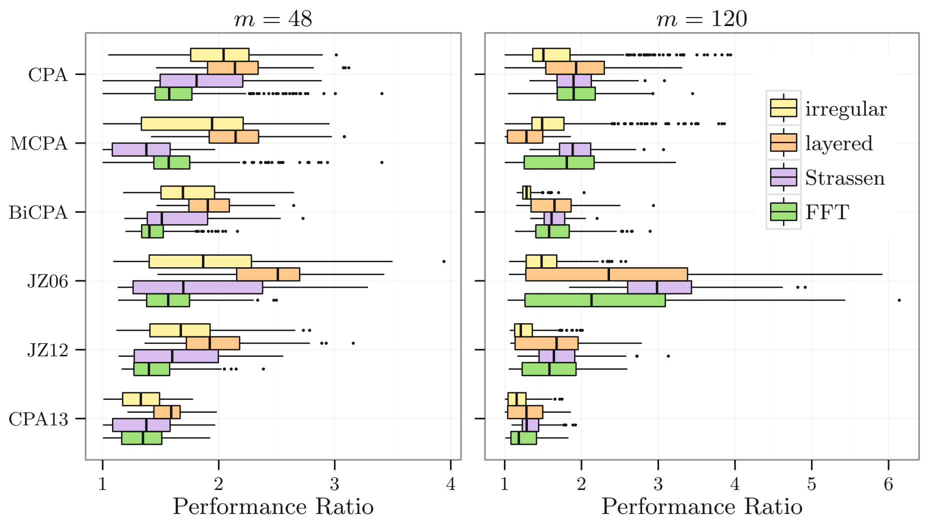


Fig. 2. Performance ratios of scheduling algorithms for each DAG class (Run-time Model 1); $m = 48$ (left) and $m = 120$ (right) processors

Figure 2 compares the performance ratio (makespan of algorithm / lower bound) of the algorithms under Run-time Model 1. The CPA13 threshold for the relative gain was set to $G = 0.01$, i.e., an allocation needs at least 1 % of runtime improvement to be considered. We can see that CPA13 achieves the lowest performance ratio, while MCPA obtains a slightly better ratio for Strassen DAGs on 48 processors and for layered DAGs on 120 processors. The reason is that CPA13 optimizes not only the makespan but also tries to keep the total work small. For a chain of parallel tasks, MCPA may allocate all processors to some task, whereas CPA13 stops if the efficiency threshold is exceeded. Thus, MCPA produces larger allocations with slightly shorter runtime, but which leads to slightly shorter schedules. The graph also shows that the performance ratio of CPA13 decreases on the bigger machine. Overall, in the cases considered, CPA13 is comparable and mostly better than JZ12, which has an approximation ratio of ≈ 3.29. We also experimented with larger DAGs, for which the linear programs of JZ06 and JZ12 have many constraints. On an Intel Core i7 (2.3 GHz) using IBM CPLEX, the time for solving instances with 1000 tasks and 120 processors took on average 49 s with JZ12 and 28 s with JZ06. In contrast, CPA13 produces solution for these instances in an average time of 2.5 s, which shows its scalability.

Run-time Model 2. The second study examines parallel tasks with arbitrary run-time functions. Here, we also include the meta-heuristic EMTS that performs an evolutionary schedule optimization [16].

Since CPA and MCPA only assume non-increasing run-time functions, we make both algorithms non-monotony-aware. To do so, we change the run-time function of a parallel task as follows: We use the run-time of the next smaller

algorithm	m	work ratio	$\sqrt{C_{\max} \cdot W}$ ratio
CPA13	120 / 48	3.88 / 2.01	2.17 / 1.75
JZ12	120 / 48	3.17 / 1.85	2.16 / 1.74
JZ06	120 / 48	5.62 / 2.45	2.94 / 2.02
BiCPA	120 / 48	2.09 / 1.62	1.84 / 1.67
MCPA	120 / 48	4.05 / 2.07	2.57 / 1.99
CPA	120 / 48	4.08 / 2.08	2.67 / 2.07

Fig. 3. Performance ratios of evaluated scheduling algorithms for several DAG classes; $m = 48$ (left) and $m = 120$ (right) processors (Run-time Model 2)

processor allocation if the run-time increases when the number of processors increases, e.g., if in the original execution time model p_j the execution time of $p_j(k) < p_j(k+1)$, we set in the modified model $\tilde{p}_j(k+1) = p_j(k)$. Then the following holds: let $1 \leq k, k' \leq m, k < k'$, so $\tilde{p}(k') \leq \tilde{p}(k)$. Yet, this newly constructed run-time function $\tilde{p}$ is neither convex in run-time nor non-decreasing in work. For this reason, we cannot apply JZ06 or JZ12 but we can use CPA and MCPA with $\tilde{p}$. We distinguish them from the original versions by appending "NM+R" to the name, where "R" stands for allotment "reduction" in the following case: after the allocation procedure of CPA+NM+R or MCPA+NM+R has finished, processor allotments may be reducible, i.e., a task v_j has k processors allotted, but $\exists k'$, $k' < k$ for which $\tilde{p}(k') = \tilde{p}(k)$. If so, we assign k' processors to task v_j since the smaller allotment is not increasing the task's run-time.

Figure 3 shows the distribution of performance ratios over all DAG classes. The chart reveals that CPA13 produces mostly schedules that are close to the lower bound with a performance ratio of less than three. EMTS is a meta-heuristic that takes allotments produced by MCPA, CPA, and CPA13 as input and attempts to optimize them. In the simulations, we instantiated an (10 + 100)-EA for EMTS, i.e., $\mu = 10$ parents and $\lambda = 100$ offspring per generation. We stopped EMTS after evaluating 10 EA generations. It is therefore not surprising that EMTS has a slightly better performance ratio than CPA13. However, we can state that CPA13 already produces very short schedules since EMTS hardly can optimize them further.

6 Discussion and Conclusions

The performance of parallel applications on current hardware depends on many factors such as deep memory hierarchies. As a result, run-time functions of a parallel program depending in the number of processors are neither non-increasing or strictly convex. Hence, we designed a scheduling algorithm for moldable tasks with precedence constraints and for arbitrary run-time functions. We identified key ingredients for producing short schedules for moldable tasks through careful investigation of different problem instances, which are: (1) force task parallelism,

(2) avoid allotments with small parallel efficiency and (3) adjust allotments to reduce idle times in the mapping phase.

We showed in a detailed simulation study that the algorithm CPA13 improves schedule not only in the case of arbitrary run-time functions but also for non-increasing run-time functions. One major contribution lies in the comparison of CPA and its variants to known approximation algorithms. Our results revealed that CPA13 generates the shortest schedules among the competitors in most cases. Yet, our results are limited to the cases studied here since CPA13 has no performance guarantee, which could be addressed in future work.

References

1. Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., Sevcik, K.C., Wong, P.: Theory and practice in parallel job scheduling. In: Feitelson, D.G., Rudolph, L. (eds.) IPPS-WS 1997 and JSSPP 1997. LNCS, vol. 1291, pp. 1–34. Springer, Heidelberg (1997)
2. Drozdowski, M.: Scheduling for Parallel Processing. Springer, London (2009)
3. Tomov, S., Nath, R., Ltaief, H., Dongarra, J.: Dense linear algebra solvers for multicore with GPU accelerators. In: HIPS Workshop, pp. 1–8 (2010)
4. Lepère, R., Trystram, D., Woeginger, G.: Approximation algorithms for scheduling malleable tasks under predence constraints. Int. J. Found. Comput. Sci. **13**(04), 613–627 (2002)
5. Jansen, K., Zhang, H.: An approximation algorithm for scheduling malleable tasks under general precedence constraints. ACM Trans. Algorithms **2**(3), 416–434 (2006)
6. Jansen, K., Zhang, H.: Scheduling malleable tasks with precedence constraints. J. Comput. Syst. Sci. **78**(1), 245–259 (2012)
7. Günther, E., König, F.G., Megow, N.: Scheduling and packing malleable and parallel tasks with precedence constraints of bounded width. J. Comb. Optim. **27**(1), 164–181 (2014)
8. van de Geijn, R.A., Watts, J.: SUMMA: scalable universal matrix multiplication algorithm. Concurr. Pract. Exp. **9**(4), 255–274 (1997)
9. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discrete Math. **5**, 287–326 (1979)
10. Leung, Y.J.T. (ed.): Handbook of Scheduling: Algorithms, Models and Performance Analysis. Chapman & Hall/CRC, Boca Raton, FL, USA (2004)
11. Skutella, M.: Approximation algorithms for the discrete time-cost tradeoff problem. Math. Oper. Res. **23**(4), 909–929 (1998)
12. Radulescu, A., van Gemund, A.: A low-cost approach towards mixed task and data parallel scheduling. In: ICPP, pp .69–76 (2001)
13. Bansal, S., Kumar, P., Singh, K.: An improved two-step algorithm for task and data parallel scheduling in distributed memory machines. Parallel Comput. **32**(10), 759–774 (2006)
14. Hunold, S.: Low-cost tuning of two-step algorithms for scheduling mixed-parallel applications onto homogeneous clusters. In: CCGrid, pp. 253–262 (2010)
15. Desprez, F., Suter, F.: A bi-criteria algorithm for scheduling parallel task graphs on clusters. In: CCGrid, pp. 243–252 (2010)

16. Hunold, S., Lepping, J.: Evolutionary scheduling of parallel tasks graphs onto homogeneous clusters. In: CLUSTER, pp. 344–352 (2011)
17. Suter, F.: DAGGEN: a synthetic task graph generator. https://github.com/frs69wq/daggen
18. Albers, S., Schröder, B.: An experimental study of online scheduling algorithms. J. Exp. Algorithmics **7**, 3 (2002)

OStrich: Fair Scheduling for Multiple Submissions

Joseph Emeras[1], Vinicius Pinheiro[2](✉), Krzysztof Rzadca[3], and Denis Trystram[4,5]

[1] Laboratoire d'Informatique de Grenoble CEA - CNRS, Grenoble, France
joseph.emeras@imag.fr
[2] Laboratory for Parallel and Distributed Computing, University of São Paulo, São Paulo, Brasil
vinicius.pinheiro@ime.usp.br
[3] Institute of Informatics, University of Warsaw, Warsaw, Poland
krzadca@mimuw.edu.pl
[4] Grenoble Institute of Technology, Grenoble, France
[5] Institut Universitaire de France, Vesoul, France
trystram@imag.fr

Abstract. Campaign Scheduling is characterized by multiple job submissions issued from multiple users over time. This model perfectly suits today's systems since most available parallel environments have multiple users sharing a common infrastructure. When scheduling individually the jobs submitted by various users, one crucial issue is to ensure *fairness*. This work presents a new fair scheduling algorithm called *OStrich* whose principle is to maintain a virtual time-sharing schedule in which the same amount of processors is assigned to each user. The completion times in the virtual schedule determine the execution order on the physical processors. Then, the campaigns are interleaved in a fair way by *OStrich*. For independent sequential jobs, we show that *OStrich* guarantees the *stretch* of a campaign to be proportional to campaign's size and the total number of users. The *stretch* is used for measuring by what factor a workload is slowed down relative to the time it takes on an unloaded system. The theoretical performance of our solution is assessed by simulating *OStrich* compared to the classical FCFS algorithm, issued from synthetic workload traces generated by two different user profiles. This is done to demonstrate how *OStrich* benefits both types of users, in contrast to FCFS.

Keywords: Job scheduling · Fairness · Job campaigns · Multi-user · Workload traces

1 Introduction

High performance computing (HPC) systems are usually shared by multiple users who compete for the usage of the processors in order to execute their jobs. Most of such systems embrace users and projects around a common infrastructure that simplifies resource management and application execution through a centralized scheduler. In the past, most users were oriented toward the maximization of the throughput but the popularization of those systems attracted other types of users. Nowadays, the users turned to the optimization of the response-time [8]. Workload of response-time users is composed of multiple submissions released sequentially over time [10,17,22,24]. For such users, the criterion of throughput is not meaningful as they are more interested in the flow time of each submission.

In this work, the problem of multiple submissions on parallel system is narrowed to the notion of *Campaign Scheduling*, introduced in [18] and analyzed under restrictive assumptions. The campaign scheduling problem models a user submission pattern commonly found in parallel systems used for scientific research: a user submits a set of jobs, analyzes the outcomes and then resubmits another set of jobs. In other words, the campaigns are sets of jobs issued from a user and they must be scheduled one after the other since the submission of a new campaign depends on the outcome of the previous one. As this pattern is an interactive process, the objective from the user point of view is to minimize the time each campaign spent in the system, namely the campaign's flow time. As soon as a campaign finishes, soon the user is ready to submit the next one.

We propose in this paper a new on-line scheduling algorithm (called *OStrich*) that explicitly maintains fairness between the users by bounding the worst-case stretch of each campaign. We show that user's performance does not depend on the total load of the system, but only on the number of users competing for the processors and on its own workload. We are also able to quantify and optimize the *performance* of each user.

The rest of this paper is organized as follows. In the next section, we give an overview of the state-of-the-art of scheduling with multiple users. In Sect. 3 we present a formal model of Campaign Scheduling. This model is an extension of what was defined in [18]. Section 4 is dedicated to the description of our solution, a new algorithm for the problem of campaign scheduling with multiple users. The theoretical results are depicted and analyzed in Sect. 5. Our solution is assessed through simulations that uses user profiles based on short and long job lengths. This is presented in Sect. 6. Finally, we present our conclusions and future work in Sect. 7.

2 State-of-the-Art

The main works related to this paper address the problem of optimizing various users criteria and the use of fair policies on parallel systems. The Multi-Users Scheduling Problem was introduced for a single processor and two users by Agnetis et al. [1]. This study focused on optimization problems. The authors show that when both users are interested each in their makespan, the problem can be solved in polynomial time. Saule and Trystram [21] extended the analysis for scheduling independent sequential jobs belonging to k different users on m identical processors. This is an offline problem where all the jobs are known in advance and can be immediately executed. They proved that the problem becomes strongly NP-hard as soon as one user aims at optimizing the makespan.

Fairness is an important issue while designing scheduling policies and it has gained growing attention in the last decade. However, it is still a fuzzy concept that has been handled in many different ways, varying according to the target problems. In [19,20], several metrics are proposed for expressing the degree of unfairness in various systems. Both works evaluate the unfairness of algorithms such as FCFS, backfilling and processor sharing, but fairness is associated with the jobs and their service requirements. Thus, the concept of fairness is seen as "fairness between jobs" instead of "fairness between users" as we propose.

There are two classical approaches to share a resource between users in a system: space sharing and time sharing. In space sharing, the resource is divided into subsets that are assigned to each party. This can be more easily applied to divisible resources such as computer memory, bandwidth and parallel systems. For indivisible resources like single processing units and I/O devices, time sharing may be a more appropriate approach, since it gives time slices to the parties in a round-robin fashion. During each time slice the resource is available to just one user.

In this work, we propose that fair schedules can be achieved through a combination of both strategies, ruled by a fair allocation metric. In the literature, one of the accepted and used metrics is the stretch, i.e. the flow time normalized by the job's processing time. The stretch and flow metrics were first studied by Bender et al. [3] for continuous job streams. Stretch optimization was also studied for independent tasks without preemption [4], Bag-of-Tasks applications [7,16], multiple parallel task graphs [6] and for sharing broadcast bandwidth between clients requests [23].

The concept of campaign is related to the bag-of-task model [2,14] and the groups of jobs model [13]. However, campaigns differ from bag-of-tasks as the jobs belonging to a single campaign have different lengths, sizes and dependencies. Unlike the group of jobs model, we assume that a user does not submit the subsequent campaign until the previous one was completed. Our scheduling algorithm uses the concept of a virtual completion time. Similar ideas are used for fair queuing in communication networks (see [11] and the references within).

3 Model and Problem Definition

The model consists of k users (indexed by u) sharing the processors on a parallel platform composed of m identical processors. The processors are managed by a centralized scheduler. A user workload is composed of successive campaigns where each campaign, indexed by i is submitted at a time denoted by t_i^u and is composed of a set of independent and non-preemptive sequential jobs. We consider an on-line problem in which any particular campaign i (and its jobs) is unknown to the scheduler until it is submitted. A campaign is defined as the set J_i^u containing the jobs released by a user u in one submission; n_i^u denotes the number of jobs of a campaign and n^u the total number of jobs released in all the campaigns of user u.

The jobs inside a campaign are indexed by j. The job's length is denoted by $p^u_{i,j}$ and, so, the total workload within campaign i is: $W^u_i = \sum_j p^u_{i,j}$. A job, once started, cannot be interrupted nor preempted. The job start time is denoted by $s^u_{i,j}$ and its completion time is denoted by $C^u_{i,j}$.

The start time of a campaign is denoted by s^u_i. It is defined as the time the first job starts, $s^u_i \triangleq \min_j s^u_{i,j}$. The campaign's completion time C^u_i is the time the last job completes, $C^u_i \triangleq \max_j C^u_{i,j}$.

The campaign's flow time, denoted as Δ^u_i, is equal to the amount of time the jobs of a campaign stay in the system: $\Delta^u_i \triangleq C^u_i - t^u_i$.

The campaign's stretch is denoted by D^u_i and is defined as a generalization of a job's stretch. Formally, a job stretch $D^u_{i,j}$ is equal to the relative degradation of its flow time, $D^u_{i,j} = (C^u_{i,j} - t^u_{i,j})/p^u_{i,j}$, where $p^u_{i,j}$ is the job length (and, thus, the job's optimum flow time) [3]. Determining a single campaign's optimum flow time means solving the general scheduling problem, so it is NP-hard. Thus, instead, we use a lower bound on campaign's flow time defined by $l^u_i = \max(W^u_i/m, p^u_{\max})$, where $p^u_{\max} = \max_j p^u_{i,j}$. Consequently, the campaign's stretch is defined as $D^u_i = \Delta^u_i / l^u_i$.

A user u cannot submit her-his next campaign $i+1$ until her-his previous campaign i completes, thus $t^u_{i+1} \geq C^u_i$. The time between the completion of campaign i and the submission of the next one $(i+1)$, called the *think time*, is denoted as $tt^u_{i+1} = t^u_{i+1} - C^u_i$.

The objective is to minimize the per-user and per-campaign stretch D^u_i. We consider stretch (in contrast to the flow time), as it weights the responsiveness of the system by the assigned load; it is natural to expect that small workloads will be computed faster than larger ones. We consider it on a per-user basis, as this results in fairness of the system towards individual users. Moreover, considering stretch of each campaign (rather than the overall stretch) guarantees that not only the final result, but also the intermediate ones are timely computed.

The problem of minimizing per-user and per-campaign stretch D^u_i is NP-hard, as when restricted to a single user ($k = 1$) and to a single campaign, it is equivalent to the classical problem of minimization of the makespan on identical parallel processors $(P||C_{\max})$ [18,21].

4 Algorithm

We propose in this section a new scheduling algorithm called *OStrich*. The algorithm guarantees the worst-case stretch of each campaign of each user D^u_i to be proportional to the campaign's workload and the number of active users in the system. OStrich's principle is to create a virtual fair-sharing schedule that determines the execution priorities of the campaigns in the real schedule. The algorithm maintains a list of ready-to-execute campaigns ordered by their priorities and interactively selects the next job from the highest priority campaign. Any scheduling policy can be used to determine the execution order of jobs within a single campaign; for instance LPT [12] (or, more appropriately, MLPT [15]) or Shortest Processing Time (SPT) [5].

The virtual fair-sharing schedule is maintained by dividing the processors between the active users at each given moment. The processors are divided *evenly* among the users, independently of users' submitted workload. The priority of a user's campaign is determined by its virtual completion time, i.e. the completion time in the virtual schedule. The campaign with the shortest virtual completion time has the priority of execution. This virtual completion time is denoted by $\tilde{C}_i^u$ for a campaign J_i^u (more generally, we will use $\tilde{x}$ for denoting a variable x in the virtual schedule). That way, if a user u submits a campaign at time t_i^u, its virtual completion time is defined as the total workload of the campaign divided by its share of processors, added by its virtual start time. More formally:

$$\tilde{C}_i^u(t) = \tilde{W}_i^u/(m/\tilde{k}(t)) + \tilde{s}_i^u = \tilde{k}(t)\tilde{W}_i^u/m + \tilde{s}_i^u. \tag{4.1}$$

Note that the share of a user is defined as the number of processors m divided by the number of active users at moment t, denoted by $\tilde{k}(t)$. By active users, we mean the users with unfinished campaigns at time t, according to the virtual schedule. Formally, $\tilde{k}(t)$ is defined as $\tilde{k}(t) = \sum_u^k \mathbb{1}\{u,t\}$ where $\mathbb{1}\{u,t\}$ is an indicating function that returns 1 if $\exists i \mid \tilde{C}_i^u > t_e$ and 0 otherwise.

A campaign starts in the virtual schedule after it is submitted, but also not sooner than the virtual completion time of the previous campaign (the previous campaign can be completed earlier in the real scheduler than in the virtual schedule). So, according to this:

$$\tilde{s}_i^u = max(t_i^u, \tilde{C}_{i-1}^u). \tag{4.2}$$

This condition guarantees that at each time moment, at most one campaign of each user is executing in the virtual schedule, as it happens on the real scheduler. Thus, the number of allocated processors depends on the number of active users, and not the system load. Additionally, *OStrich* does not allow a campaign to start before its virtual start time ($s_i^u \geq \tilde{s}_i^u$). This mechanism keeps the real schedule in accordance with the fair principles of the virtual schedule: a user is not able to take a greater share of the processors than what is assigned in the virtual schedule.

The virtual completion time of the campaigns can be updated on two events: the submission of a new campaign and the completion of a campaign in the virtual schedule. These events may change the number of active users $\tilde{k}(t)$ and, thus, modify the virtual completion times of other active campaigns. Besides, at each event e occurring at time t_e, the virtual workload of a campaign ($\tilde{W}_i^u$) must be redefined based on how much it is left to be executed in the virtual schedule. The remaining workload of a campaign is defined by taking the time passed since the last event occurrence t_{e-1} and multiplying it by campaign's share of processors on that time interval. Considering all the events passed after the campaign's submission, the workload formula is $\tilde{W}_i^u = \sum_j p_{i,j}^u - \sum_e (m.(t_e - t_{e-1})/\tilde{k}(t_{e-1}))$.

5 Theoretical Analysis

In this section, the worst case stretch of a campaign is to analyzed. The idea for the proof is to bound the completion time of the last job of a campaign using a "global area" argument compared to the virtual schedule. In this proof, $p_{\max}$ denotes the maximum job length in the system. "Area" is a measure in terms of amount of time $\times$ number of processors. The virtual schedule is denoted by V and the real schedule by R. To simplify the formulation of proofs, we will say that the virtual schedule V "executes" jobs, even though V is just an abstraction used for prioritizing real jobs. At time t, a job is "executed" by V if in V there is a fraction of processors assigned to this job.

5.1 Worst-Case Bound

As V can assign a job an arbitrary fraction of processors (from ϵ to m), a schedule in V is represented by horizontal load streams, one for each active user. Idle intervals can be present in V only when there are no ready jobs to be executed. In turn, R must execute each job on a single processor, thus some processors can be idle even when there are ready jobs. This can happen when $t_i^u < \tilde{s}_i^u$ and the ready jobs of campaign i must wait until moment $\tilde{s}_i^u$ arrives. So, the question is whether the idle times that might additionally appear in R can cause a systematic delay of R compared to V. The following lemma shows that once R is delayed by an area of $mp_{\max}$, the delay does not grow further, as there is always a ready job to be executed.

The lemma considers only the idle time in the "middle" of the schedule, i.e., after the start time of the first job and up to the start time of the last job; this is sufficient to characterize the on-line behavior of *OStrich*.

Lemma 1. *The total idle area in R (counted from the earliest to the latest job start time) exceeds the total idle area in V by at most* $mp_{\max}$.

Proof. Consider first a V schedule with no idle times. Assume by contradiction that t is the first time moment when the total idle area in R starts to exceed $mp_{\max}$. Thus, at least one processor is free at time t and there is no ready jobs to be executed. As V has no idle times, at time t the area executed by V exceeds the area executed by R by more than $mp_{\max}$. Thus, the area exceeding $mp_{\max}$ is ready to be executed at R: as a single job has an area of at most $p_{\max}$, an area of $mp_{\max}$ is composed of at least m jobs. Thus, at least m jobs are being executed, or ready to be executed in R. This contradicts the assumption that there is at least one free processor at R at time t.

If there is idle time in V, each idle period can be modeled as a set of *dummy* jobs $\{J_I\}$ that are executed by V, but not necessarily (and/or not completely) by R. If R executes $\{J_I\}$ entirely, the thesis is true by the argument from the previous paragraph (as $\{J_I\}$ contributes with the same amount $\sum p_I$ of idle area to V and to R). If R executes $\{J_I\}$ partially (as $\{J_I'\}$, with $0 \leq p_I' \leq p_I$) the contribution of these jobs to the idle area of R ($\sum p_I'$) is smaller than to V ($\sum p_I$). ■

R starts to execute jobs from campaign J_i^u when this campaign has the shortest completion time in V. Yet, it is possible that after some, but not all, jobs from J_i^u have started, another user v submits her/his campaign $J_j^{(v)}$ having a lower area than what remains of J_i^u, and thus gaining higher priority. Thus, J_i^u is executed in R in so-called *pieces*: two jobs $J_k, J_l \in J_i^u$ belong to the same piece iff no job from other campaign $J_j^{(v)}$ starts between them ($\nexists J' : J \in J_j^{(v)} \wedge s_{Jk} < s_{J'} < s_{Jl}$).

The following lemma bounds the completion time of the last piece of the campaign. After a campaign is completed in the virtual schedule, it cannot be delayed by any other newly-submitted campaign; thus it has the highest priority and its remaining jobs are executed in one piece (i.e., the last piece). The lemma upper-bounds the virtual area having higher priority by the area of the campaign, as in the worst case k users submit campaigns of equal area, thus ending at the same time in V, and thus being executed in arbitrary order in R.

Lemma 2. *The completion time $C_{i,q}^u$ of the last piece q of a campaign J_i^u is bounded by a sum:*

$$C_{i,q}^u \leq t_i^u + k\frac{W_{i-1}^u}{m} + p_{\max} + (k-1)\frac{W_i^u}{m} + p_{\max} + \frac{W_i^u}{m} + p_{\max}^u. \tag{5.1}$$

Proof. In (5.1), $t_i^u + kW_{i-1}^u/m$ expresses the maximum time a campaign may wait until the virtual completion time of the previous campaign J_{i-1}^u of the same user; $(k-1)W_i^u/m$ bounds the time needed to execute other users' campaigns that can have higher priority; $W_i^u/m + p_{\max}^u$ bounds the execution time of the campaign J_i^u; and two $p_{\max}$ elements represent the maximum lateness of R compared to V and the maximum time needed to claim all the processors.

From (4.1), (4.2) and knowing that $t_i^u \geq \tilde{s}_{i-1}^u$ (the next campaign cannot be released before the previous one starts), $\tilde{s}_i^u \leq t_i^u + kW_{i-1}^u/m$.

There is no idle time in R in the period $[\tilde{s}_i^u, s_{i,q}^u)$, otherwise, the last piece could have been started earlier.

We denote by A the area of jobs executed in R after the time moment $\tilde{s}_i^u$ and until $\tilde{s}_{i,q}^u$. We claim that $A \leq mp_{\max} + (k-1)W_i^u + W_i'^u$, where $W_i'^u$ is the area of jobs from campaign J_i^u executed until $s_{i,q}^u$. The Fig. 1 facilitates the visualization of these notations, including the area A (shaded area).

To prove the claim, we analyze job J executed in R in the period $[\tilde{s}_i^u, s_{i,q}^u)$. First, J is not executed in V after $s_{i,q}^u$. If J is in V after $s_{i,q}^u$, J has lower priority than jobs from campaign J_i^u, so *OStrich* would not choose J over jobs from campaign J_i^u.

Second, if J is executed in V before $\tilde{s}_i^u$, it means that R is "late" in terms of executed area: but the total area of such "late" jobs it at most $mp_{\max}$ (from Lemma 1).

Thus, if J has a corresponding area in the virtual schedule executed in the period $[\tilde{s}_i^u, s_{i,q}^u)$, the area A of the jobs started in the real schedule in this period is equal to the area of the virtual schedule between $[\tilde{s}_i^u, s_{i,q}^u)$ plus the lateness $mp_{\max}$. Recall that from time $s_{i,q}^u$ till the start of the last job of J_i^u, the campaign

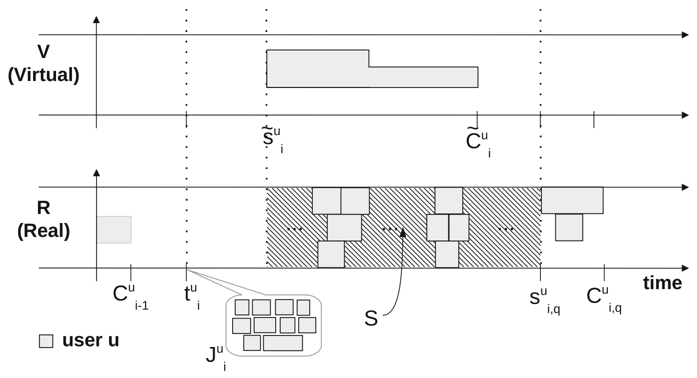


Fig. 1. Analysis of *OStrich*: bound for the campaign stretch

J^u_i has the highest priority (as it is not interrupted by any other campaign). Thus, at the latest, $s^u_{i,q}$ corresponds to the time moment $\tilde{C}^u_i$ in the virtual schedule when the campaign J^u_i completes (plus the lateness $p_{\max}$). Thus, by definition of the virtual schedule, $s^u_{i,q} \leq p_{\max} + \tilde{s}^u_i + k\frac{W^u_i}{m}$.

Starting from $s^u_{i,q}$, the remaining jobs of J^u_i start and complete. J^u_i can claim all processors at the latest $p_{\max}$ after $s^u_{i,q}$. Then, by using classic lower bounds, it takes $W^u_i/m + p^u_{\max}$ to complete the campaign. ■

The following theorem states that in *OStrich* the stretch of a campaign depends only on the number of active users and the relative area of two consecutive campaigns.

Theorem 1. *The stretch of a campaign is proportional to the number of active users k*[1] *and the relative area of consecutive campaigns.* $D^u_i \in O(k(1 + \frac{W^u_{i-1}}{W^u_i}))$.

Proof. The result follows directly from Lemma 2. Recall that, for campaign J^u_i, the stretch D^u_i is defined by $D^u_i = (C^u_i - t^u_i)/l^u_i = (C^u_i - t^u_i)/\max(W^u_i/m, p^u_{\max})$. Also, $C^u_i = C^u_{i,q}$, i.e. the completion time of a campaign is equal to the completion time of its last "piece". Replacing C^u_i by the definition of $C^u_{i,q}$ taken from Lemma 2, we have

$$D^u_i \leq \frac{\frac{kW^u_{i-1}}{m} + 3p_{\max} + \frac{kW^u_i}{m}}{\max(\frac{W^u_i}{m}, p^u_{\max})} \leq k(1 + \frac{W^u_{i-1}}{W^u_i}) + 3mp_{\max}.$$

For a given supercomputer, m is constant; similarly, the maximum size of a job $p_{\max}$ can be treated as constant, as it is typically limited by system administrators. Hence, $D^u_i \in O(k(1 + W^u_{i-1}/W^u_i))$. ■

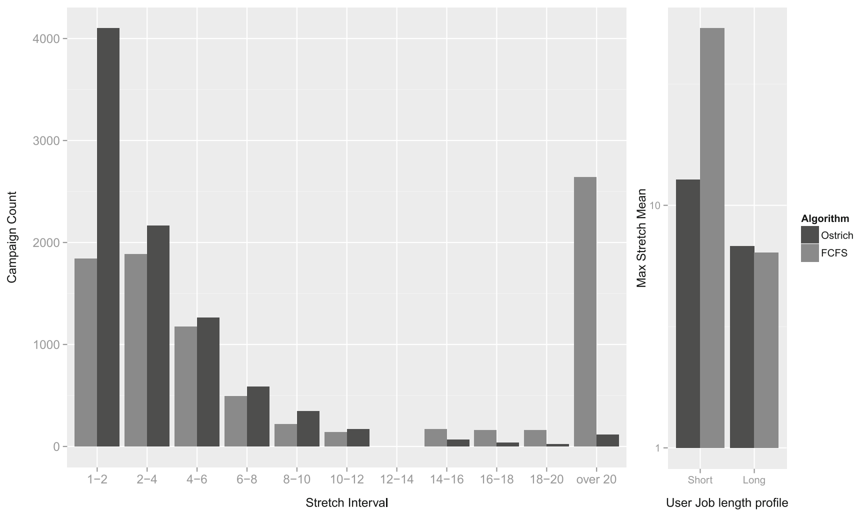
[1] The number of active users may vary on time. Here, we assume that k is the biggest value it assume during the execution of the campaign.

It is worth noting that the stretch does not dependent on the current total load of the system. Heavily-loaded users do not influence the stretch of less-loaded ones. Also, this bound is tight (the proof is available at [9]).

6 Simulations

In this section, we analyze the performance of *OStrich*. We present a simulation to demonstrates that *OStrich* results in lower stretch values against. The results produced by *OStrich* are compared with a FCFS classical algorithm that schedules campaigns in a First-In-First-Out order. Despite its simplicity, we chose FCFS because it is a widely used strategy, easy to understand and to implement and jobs do not starve. The simulator plays the role of a centralized scheduler: it takes an instance of a user workload as inputs; and it calculates the campaign stretches and the max-stretch per user obtained by each algorithm in an environment composed of $m = 64$ identical processors.

We run 40 instances where instance is composed of 10^4 jobs. For each job we set its length p according to the user profile. In order to observe how different user profiles are affected by each algorithm, we defined 2 user profiles: *short* and *long*. Short users submit short jobs with lengths uniformly taken from the range $[1; 3.6 \times 10^3]$ (seconds as time unit). Long users submit long jobs with lengths uniformly taken from the range $[3.6 \times 10^3; 3.6 \times 10^4]$. Each job starts a new campaign with probability of 0.02; otherwise, it belongs to the previous campaign. If the job starts a new campaign, we set the owner of this campaign according to a uniform distribution.



(a) Stretch values by intervals (b) Max campaign stretch per user

Fig. 2. Ostrich vs FCFS: comparing stretch values of campaigns

In general, the results confirm our expectations and show that *OStrich* results in significantly lower max stretches than FCFS. The Fig. 2a shows the distribution of stretch values for all campaigns. The number of campaigns with stretch lower than 2 for *OStrich* is more than twice the number obtained with FCFS. More important, though is the number of high stretch values above: while on *OStrich* this number decreases rapidly towards 0 as stretch increases, with FCFS it is bigger than 2600 above 20, representing 42.3 % of the total. The occurrence of stretch values above 20 is only 117 for *OStrich* (1.3 %).

The Fig. 2b shows the max stretch average per user profile (in a log scale) and here we can see how *OStrich* accomplishes its purpose: short users are penalized by FCFS with big stretch values (whose average is above 50) while *OStrich* does not heavily discriminate users by their profiles, guaranteeing a more fair treatment for all the users (average of 12.8 for short users). For long users, FCFS and *OStrich* have almost the same performance (average of 6.3 for FCFS and 6.8 for *OStrich*).

7 Concluding Remarks

We have presented in this work a new scheduling algorithm for the problem of scheduling multiple submissions issued by many users. *OStrich* algorithm has been designed to handle the problem of fairness between users by defining execution priorities according to a criterion based on *stretch*. The principle of the proposed solution is to dynamically determine the priorities between the campaigns based on a fair time-sharing virtual schedule. We proved that *OStrich* delivers performance guarantee for the max stretch value of a user campaign that depends only on the user workload and on the number of active users.

The performance of our algorithm is assessed by running simulations on workloads composed of two types of users sharing a parallel system. The results show that, for short job user profiles, *OStrich* achieves lower stretches than the FCFS while it remains as good as FCFS for long job users; moreover distribution of stretches among users is more equal. Therefore, *OStrich* delivers a good compromise between fairness and user performance without worsening overall performance.

Acknowledgement. Krzysztof Rzadca is partly supported by Polish National Research Center SONATA grant UMO-2012/07/D/ST6/02440 Work partly supported by the French-Polish scientific cooperation program POLONIUM. Vinicius Pinheiro is partly supported by the CAPES/COFECUB Program (project number 4971/11-6).

References

1. Agnetis, A., Mirchandani, P.B., Pacciarelli, D., Pacifici, A.: Scheduling problems with two competing agents. Oper. Res. **52**(2), 229–242 (2004)
2. Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Marchal, L., Robert, Y.: Centralized versus distributed schedulers for bag-of-tasks applications. IEEE Trans. Parallel Distrib. Syst. **19**(5), 698–709 (2008)

3. Bender, M.A., Chakrabarti, S., Muthukrishnan, S.: Flow and stretch metrics for scheduling continuous job streams. In: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'98, pp. 270–279. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1998). http://dl.acm.org/citation.cfm?id=314613.314715
4. Bender, M.A., Muthukrishnan, S., Rajaraman, R.: Improved algorithms for stretch scheduling. In: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'02, pp. 762–771. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2002). http://dl.acm.org/citation.cfm?id=545381.545482
5. Bruno, J., Coffman, J.E.G., Sethi, R.: Scheduling independent tasks to reduce mean finishing time. Commun. ACM **17**(7), 382–387 (1974). http://doi.acm.org/10.1145/361011.361064
6. Casanova, H., Desprez, F., Suter, F.: Minimizing stretch and makespan of multiple parallel task graphs via malleable allocations. In: 2010 39th International Conference on Parallel Processing (ICPP), September 2010, pp. 71–80 (2010)
7. Celaya, J., Marchal, L.: A fair decentralized scheduler for bag-of-tasks applications on desktop grids. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), May 2010, pp. 538–541 (2010)
8. Donassolo, B., Legrand, A., Geyer, C.: Non-cooperative scheduling considered harmful in collaborative volunteer computing environments. In: Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid, Computing, CCGRID'11, pp. 144–153 (2011)
9. Emeras, J., Pinheiro, V., Rzadca, K., Trystram, D.: Fair scheduling for multiple submissions. Research Report RR-LIG-033, LIG, Grenoble, France (2012)
10. Feitelson, D.: Workload modeling for computer systems performance evaluation (2005). http://www.cs.huji.ac.il/feit/wlmod/wlmod.pdf
11. Ghodsi, A., Sekar, V., Zaharia, M., Stoica, I.: Multi-resource fair queueing for packet processing. ACM SIGCOMM Comput. Commun. Rev. **42**(4), 1–12 (2012)
12. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM JOURNAL ON APPLIED MATHEMATICS **17**(2), 416–429 (1969)
13. Iosup, A., Jan, M., Sonmez, O.O., Epema, D.H.J.: The Characteristics and Performance of Groups of Jobs in Grids. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 382–393. Springer, Heidelberg (2007)
14. Iosup, A., Sonmez, O., Anoep, S., Epema, D.: The performance of bags-of-tasks in large-scale distributed systems. In: Proceedings of the 17th International Symposium on High Performance Distributed Computing, pp. 97–108. ACM (2008)
15. Lee, C.Y.: Parallel machines scheduling with nonsimultaneous machine available time. Discrete Appl. Math. **30**, 53–61 (1991). http://dx.doi.org/10.1016/0166-218X(91)90013-M
16. Legrand, A., Su, A., Vivien, F.: Minimizing the stretch when scheduling flows of biological requests. In: Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA'06, pp. 103–112. ACM, New York, NY, USA (2006). http://doi.acm.org/10.1145/1148109.1148124
17. Mehrzadi, D., Feitelson, D.G.: On extracting session data from activity logs. In: Proceedings of the 5th Annual International Systems and Storage Conference, SYSTOR'12, pp. 3:1–3:7 (2012)
18. Pinheiro, V., Rzadca, K., Trystram, D.: Campaign scheduling. In: IEEE International Conference on High Performance Computing (HiPC), Proceedings (2012, accepted for publication)

19. Raz, D., Levy, H., Avi-Itzhak, B.: A resource-allocation queueing fairness measure. SIGMETRICS Perform. Eval. Rev. **32**(1), 130–141 (2004)
20. Sabin, G., Kochhar, G., Sadayappan, P.: Job fairness in non-preemptive job scheduling. In: Proceedings of the 2004 International Conference on Parallel Processing, ICPP'04, pp. 186–194 (2004)
21. Saule, E., Trystram, D.: Multi-users scheduling in parallel systems. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium, May 2009, pp. 1–9. Washington, DC, USA (2009)
22. Shmueli, E., Feitelson, D.: Using site-level modeling to evaluate the performance of parallel system schedulers. In: 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006, MASCOTS 2006, September 2006, pp. 167–178 (2006)
23. Wu, Y., Cao, G.: Stretch-optimal scheduling for on-demand data broadcasts. In: Proceedings of Tenth International Conference on Computer Communications and Networks, pp. 500–504 (2001)
24. Zakay, N., Feitelson, D.G.: On identifying user session boundaries in parallel workload logs. In: Proceedings of the 16th Workshop on Job Scheduling Strategies for Parallel Processing. The Hebrew University, Israel (May 2012). http://www.cs.huji.ac.il/feit/parsched/jsspp12/p12-zakay.pdf

Fair Share Is Not Enough: Measuring Fairness in Scheduling with Cooperative Game Theory

Piotr Skowron(✉) and Krzysztof Rzadca

Faculty of Mathematics, Informatics and Mechanics,
University of Warsaw, Warsaw, Poland
{p.skowron,krzadca}@mimuw.edu.pl

Abstract. We consider the problem of fair scheduling in a multi-organizational system in which organizations contribute their own resources to the global pool and the jobs to be processed on the common resources. We consider on-line, non-clairvoyant scheduling of sequential jobs without preemption. To ensure that the organizations are willing to cooperate the scheduling algorithm must be fair.

To characterize fairness, we use a cooperative game theory approach. The contribution of an organization is computed based on how this organization influences the utility (which can be any metric, e.g., flow time, turnaround, resource allocation) of all organizations. Formally, the contribution of the organization is its Shapley value in the cooperative game. The scheduling algorithm should ensure that the contributions of the organizations are close to their utilities. Our previous work proves that this problem is NP-hard and hard to approximate.

In this paper we propose a heuristic scheduling algorithm for the fair scheduling problem. We experimentally evaluate the heuristic and compare its fairness to fair share, round robin and the exact exponential algorithm. Our results show that fairness of the heuristic algorithm is close to the optimal. The difference between our heuristic and the fair share algorithm is more visible on longer traces with more organizations. These results show that assigning static target shares (as in the fair share algorithm) is not fair in multi-organizational systems and that instead dynamic measures of organizations' contributions should be used.

Keywords: Fair scheduling · Game theory · Algorithm

1 Introduction

A large fraction of contemporary supercomputing resources is run by consortia of independent organizations: from local supercomputing centers shared by a few research groups to international grids, such as Grid5000, DAS or Planet-lab. Each participating organization grants access to its resources to other members of the consortium; in return, an organization expects to be given a fair access to other resources. The role of the consortium is to coordinate access to the resources through a scheduler. Fairness is crucial to the existence of such systems: if an organization feels that it is treated unfairly, it may quit the consortium, thus reducing the pool of the resources accessible by others.

Fairness is one of the key problems in scheduling. Most existing approaches [2,6–10,18] are based on distributive fairness: agents (users, projects or organizations) are assigned a target share of the available resources. The scheduler's goal is to produce schedules with an average utilization per agent close to the target share. However, distributive fairness does not correspond with agents' goals: an agent, rather than being given an equal share of resources, wants its jobs to be completed fast. Alternative approaches consider agents' utilities (and, sometimes, possible actions of agents). Kostreva et al. [11] proposes an axiomatic characterization of fairness based on multi-objective optimization; Rzadca et al. [16] applies this concept to scheduling in a multi-organizational system. [3,4] optimizes the global makespan with an additional constraint that each organization must be at least as well-off as if it acted alone. See [17] for more detailed related work.

In our previous work [17], we considered the problem of fairness by determining the Shapley value of each organization (we summarize these results in Sect. 3). The Shapley value is a concept commonly used in the cooperative game theory. For an organization, its Shapley value expresses the relative value of the organization to the others. Thus, it represents a fair amount of utility an organization should get from the schedule. In contrast to other works [1,12–14] using monetary valuations for jobs, we proposed to compute the Shapley value directly as a function of how organization's processors increase other organization utilities; and how organization's jobs decrease others utilities. The problem is that an exact scheduling algorithm that produces schedules approximating the Shapley value is exponential ($O(3^n)$).

The contribution of this paper is the following. First, we propose a practical heuristic that schedules jobs according to an estimated Shapley value (Algorithm DirectContr in Sect. 4). The heuristic estimates the contribution of an organization by the number of CPU-timeunits an organization contributes for computing jobs of other organizations; the algorithm schedules the jobs to minimize the maximal difference between the utility and the contribution over all organizations. Second, we conduct extensive simulation experiments to verify fairness of commonly-used scheduling algorithms (Sect. 5). The experiments show that although the fair share algorithm is considerably better than round robin (which does not aim to optimize fairness), our heuristic constantly outperforms fair share, being close to the optimal algorithm and the randomized approximation algorithm. The main conclusion is that ensuring that each party is given a fair share of resources (the distributive fairness approach) might not be sufficient in systems with dynamic job arrival patterns. An algorithm based on the Shapley value, that explicitly considers the organization's impact on other organizations' utilities, produces more fair schedules.

2 The Scheduling Model

We consider a multi-organizational model in which the organizations *may* cooperate; the set of cooperating organizations is called a *coalition* and denoted as $\mathcal{C}$. Every subset of a coalition is also a coalition; however, to emphasize that we are considering a subset of the organizations from a particular $\mathcal{C}$, we will refer to such subsets as to *subcoalitions* of $\mathcal{C}$. Each organization $O^{(u)}$ participating in a coalition $\mathcal{C}$ is contributing its local resources (its processors) to the coalition's global pool. In return, each organization from $\mathcal{C}$ can use all the processors from the coalition's pool to process its own jobs.

The jobs of the organizations may compete for the processors (this happens when in a given time moment there are more jobs waiting for execution than the total number of free processors), so the organizations need to agree on the scheduling algorithm. Each organization wants its jobs to be processed as fast as possible. The satisfaction of an organization from a schedule can be quantified by a utility function. The utility function of the organization can be any metric that depends on the completion times of the jobs owned by this organization. The classic utilities in the scheduling theory are: flow time, tardiness, turnaround, resource utilization, etc. Hereinafter we will use ψ when referring to the utility function.

The total utility of the organizations participating in coalition $\mathcal{C}$ is called the *value of the coalition* and denoted as $v(\mathcal{C})$. Thus, $v(\mathcal{C}) = \sum_{O^{(u)} \in \mathcal{C}} \psi(O^{(u)})$.

We consider on-line, non-clairvoyant scheduling of sequential jobs. The started jobs cannot be stopped, canceled, or preempted. The organizations decide about the order of processing their own jobs: the jobs of a single organization must be executed in the order they were presented by the organization. The processors are identical.

3 Fairness by the Shapley Value

In this section we describe our approach to fair scheduling by computing the Shapley value. The section summarizes the theoretical results from our previous work [17].

3.1 Computing the Shapley Value

The core idea of our approach lies in computing the effective influence an organization has on the utility of other organizations. The standard, budget-based approaches, when computing the load of an organization, just compute the number of CPU-seconds consumed by the jobs belonging to the organization. This approach ignores the fact that the resources used in peak load periods should be comparatively more expensive than the resources used during low load periods. Similarly, resources contributed by an organization are more valuable during peak load times than when other resources are already idle. Their value directly stems from their ability to execute waiting jobs and thus improve the overall performance.

Taking this approach, the *marginal contribution* of the organization $O^{(u)}$ to a coalition $\mathcal{C}$ ($O^{(u)} \notin \mathcal{C}$) is $v(\mathcal{C} \cup \{O^{(u)}\}) - v(\mathcal{C})$), i.e., the difference of the total utility of organizations belonging to $\mathcal{C}$ (including $O^{(u)}$) when $O^{(u)}$ joins $\mathcal{C}$. Intuitively, the marginal contribution of the organization $O^{(u)}$ to a coalition $\mathcal{C}$ measures how the presence of the organization $O^{(u)}$ influences (increases or decreases) the completion times of the jobs (the utility) of all organizations participating in $\mathcal{C}$.

The *contribution* $\phi^{(u)}(\mathcal{C})$ of the organization $O^{(u)}$ is its Shapley value. Intuitively, the Shapley value expresses the relative worth of an organization. Formally, let $\mathcal{L}_{\mathcal{C}}$ denote all orderings of the organizations from the coalition $\mathcal{C}$. Each ordering $\prec_{\mathcal{C}}$ can be associated with a permutation of the set $\mathcal{C}$, thus $|\mathcal{L}_{\mathcal{C}}| = |\mathcal{C}|!$. For the ordering $\prec_{\mathcal{C}} \in \mathcal{L}_C$ we define $\prec_{\mathcal{C}} (O^{(i)}) = \{O^{(j)} \in \mathcal{C} : O^{(j)} \prec_{\mathcal{C}} O^{(i)}\}$ as the set of all organizations from $\mathcal{C}$ that precede $O^{(i)}$ in the order $\prec_{\mathcal{C}}$. The Shapley value can be expressed [15] in the following form:

$$\phi^{(u)}(v(\mathcal{C})) = \frac{1}{|\mathcal{C}|!} \sum_{\prec_{\mathcal{C}} \in \mathcal{L}_{\mathcal{C}}} \Big(v(\prec_{\mathcal{C}} (O^{(u)}) \cup \{O^{(u)}\}) - v(\prec_{\mathcal{C}} (O^{(u)})\Big) . \qquad (1)$$

When computing the contribution $\phi^{(u)}(\mathcal{C})$ of the organization $O^{(u)}$ to a coalition $\mathcal{C}$ we consider the process of formation of $\mathcal{C}$—we consider that the organizations may join $\mathcal{C}$ in different orders. For each such an order $\prec$, the organization $O^{(u)}$ joins some already formed subcoalition $\mathcal{C}' \subset \mathcal{C}$. This subcoalition $\mathcal{C}'$ consists of the organizations that joined before $O^{(u)}$, and so of the organizations that are before $O^{(u)}$ in $\prec$. The organization $O^{(u)}$ joining $\mathcal{C}' \subset \mathcal{C}$ changes the value of the coalition by $v(\mathcal{C}' \cup \{O^{(u)}\}) - v(\mathcal{C}')$) (this is the marginal contribution of $O^{(u)}$ to $\mathcal{C}'$). The contribution of the organization $O^{(u)}$ to a coalition $\mathcal{C}$ is the expected marginal contribution of $O^{(u)}$ when the expectation is taken over all orders of the organizations from $\mathcal{C}$.

An ideally-fair scheduling algorithm should ensure that for each organization $O^{(u)}$, its utility is equal to its contribution, $\forall_u \ \psi(O^{(u)}) = \phi^{(u)}(v(\mathcal{C}))$; however, as the scheduling problem is discrete, such a solution may not exist. Instead, the goal should be to construct in each time moment a schedule that is as fair as possible. More formally, an on-line scheduling algorithm, when there is a free processor, should choose a job of an organization $O^{(u)*}$ that, after being scheduled, minimizes the distance of contributions to utilities, $|\sum_u \psi(O^{(u)}) - \phi^{(u)}(v(\mathcal{C}))|$.

The problem is that, in order to compute contribution $\phi^{(u)}(v(\mathcal{C}))$, each of $2^{|\mathcal{C}|}$ possible coalitions must be analyzed. The complexity of the resulting scheduling algorithm is $O(|\mathcal{O}|(2^{|\mathcal{O}|} \sum m^{(u)} + 3^{|\mathcal{O}|}))$ [17].

3.2 Strategy-Resilient Utility Functions

When defining fairness we need to compare values of utility functions. In distributive fairness the utilities of the organizations should be proportional to their weights. In Shapley fairness (Sect. 3.1) the utilities should be close to the contributions. However, most of the classic utility functions create incentives for organizations to manipulate their workload. An organization can change its utility by e.g. merging, splitting or delaying their jobs. E.g., consider a job released and started in time 0 and completed in time 2. The flow time of the job is 2. If the job is split into two smaller jobs – one started in time 0 and completed in time 1 and the other started in time 1 and completed in time 2, then the total flow time of the two jobs is 3. Thus, by splitting a job, the organization can later require better service by claiming that it obtained worse service from what it actually got.

A strategy-resilient (non-manipulable) utility function exists [17]. Let σ denote a schedule of the jobs of a given organization. We assume that σ is a set of pairs (s, p), each pair representing a single job; s denotes the start time and p denotes the processing time of a job. If the job is not yet completed (p is not known), we set $p = (t - s)$. A strategy-resilient utility function in time t has the following form:

$$\psi_{sp}(\sigma, t) = \sum_{(s,p)\in\sigma : s \le t} p\left(t - \frac{s + (s + p - 1)}{2}\right). \qquad (2)$$

Intuitively, in ψ_{sp} the jobs are considered as sets of unit-size tasks. Each task obtains a utility proportional to the time in which it completes. If a job completes at time t_c, at time t it gets a utility equal to $(t - t_c)$. This function can be thought of as the throughput of the jobs of an organization. If we consider a fixed set of jobs with equal processing times, maximization of ψ_{sp} is equivalent to minimization of the flow time [17]. The utility function ψ_{sp} takes into account only the completed jobs and the completed unit-size parts of the jobs (by setting $p = (t - s)$ whenever job is still being processed), thus it is adequate for non-clairvoyant models).

4 Algorithms

In this section we describe the algorithms that we evaluate. We start by a description of the exact exponential fair algorithm Ref [17]; we then describe a randomized algorithm Rand [17] approximating Ref; our new heuristic DirectContr; and the reference algorithms RoundRobin and FairShare in three versions differing by what the algorithm balance: the shares of assigned processors in FairShare, the utility functions in UtFairShare and the number of concurrently executed jobs in CurrFairShare.

Ref. This algorithm is a direct implementation of the definition from Sect. 3.1. It is based on the concepts of utilities and contributions of the organizations. The contribution of the organization is defined by its Shapley value.

The referral algorithm schedules the jobs to ensure that the contributions of organizations are as close to their utilities as possible. Calculating the Shapley value for the organization is NP-hard and hard to approximate [17]. The core difficulty lies in the fact that to calculate the Shapley value in each time moment one has to know the schedules for each subset—with k organizations there are

2^k such subsets. As the result, the practical usage of the algorithm Ref is limited and we can only use this algorithm as a benchmark for evaluating other algorithms.

Rand. Taking into account the computational hardness of the algorithm Ref, in our previous work [17] we proposed a randomized algorithm. Instead of remembering the schedules for all 2^k subsets, the algorithm for each organization $O^{(u)}$ selects only N random subsets not containing $O^{(u)}$. The contribution of $O^{(u)}$ is calculated based on the marginal contribution of $O^{(u)}$ only to these N subsets (the idea is similar to Monte-Carlo methods for computing the Shapley value). Such approach guarantees that for sufficiently large N with high probability we can get arbitrarily good approximation bounds for the fairness [17]. Although this algorithm has good theoretical properties, it requires a large N to produce high-quality schedules.

DirectContr (see Algorithm 1)**.** The algorithm keeps for each organization O its utility $\psi_{sp}[O]$ and its estimated contribution $\phi[O]$. The estimate of the

Algorithm 1. DirectContr: a heuristic algorithm for Shapley-fair scheduling.

```
Notation:
own(M), own(J) — the organization owning the processor $M$, the job $J$
$wait(O)$ — the set of released, but not-yet scheduled jobs of the organization $O$
at time $t$
Initialize($\mathcal{C}$):
    foreach $O^{(u)} \in \mathcal{C}$ do
        $\text{finUt}[O^{(u)}] \leftarrow 0$; $\text{finCon}[O^{(u)}] \leftarrow 0$ ;
        $\phi[O^{(u)}] \leftarrow 0$; $\psi[O^{(u)}] \leftarrow 0$ ;
Schedule($t_{prev}, t$): // $t_{prev}$ is the time of the previous event
    foreach $O^{(u)} \in \mathcal{C}$ do
        $\phi[O^{(u)}] \leftarrow \phi[O^{(u)}] + (t - t_{prev})\text{finCon}[O^{(u)}]$;
        $\psi[O^{(u)}] \leftarrow \psi[O^{(u)}] + (t - t_{prev})\text{finUt}[O^{(u)}]$;
    $\gamma \leftarrow$ generate a random permutation of the set of all processors;
    foreach $m \in \gamma$ do
        if not FreeMachine($m$, $t$) then
            $J \leftarrow$ RunningJob($m$);
            $\text{finUt}[\texttt{own}(J)] \leftarrow \text{finUt}[\texttt{own}(J)] + t - t_{prev}$ ;
            $\text{finCon}[\texttt{own}(m)] \leftarrow \text{finCon}[\texttt{own}(m)] + t - t_{prev}$ ;
            $\phi[\texttt{own}(J)] \leftarrow \phi[\texttt{own}(J)] + \frac{1}{2}(t - t_{prev})(t - t_{prev} + 1)$;
            $\psi[\texttt{own}(m)] \leftarrow \psi[\texttt{own}(m)] + \frac{1}{2}(t - t_{prev})(t - t_{prev} + 1)$;
    foreach $m \in \gamma$ do
        if FreeMachine($m$, $t$) and $\bigcup_{O^{(u)}} wait(O^{(u)}) \neq \emptyset$ then
            $org \leftarrow \text{argmax}_{O^{(u)}:wait(O^{(u)})\neq\emptyset}(\phi[O^{(u)}] - \psi[O^{(u)}])$ ;
            $J \leftarrow$ first waiting job of $org$ ;
            startJob($J$, $m$) ;
            $\text{finUt}[org] \leftarrow \text{finUt}[org] + 1$ ;
            $\text{finCon}[\texttt{own}(m)] \leftarrow \text{finCon}[\texttt{own}(m)] + 1$ ;
```

contribution of each organization is assessed directly (without considering any subcoalitions) by the following heuristic. On each scheduling event t we consider the processors in a random order and assign waiting jobs to free processors. The job that is started on processor m increases the contribution $\tilde{\phi}$ of the owner of m by the utility of this job.

In the pseudo code, finUt$[O]$ denotes the number of the unit-size parts of the jobs of the organization O that are completed before t_{prev}. From Eq. 2 we know that the utility in time t of the unit-size parts of the jobs of the organization O that are completed before t_{prev} is greater by $(t - t_{prev})$finUt$[O]$ than this utility in time t_{prev} (line 1); the utility of the unit-size parts of the job completed between t_{prev} and t is equal to $\sum_{i=1}^{t-t_{prev}} i = \frac{1}{2}(t - t_{prev})(t - t_{prev} + 1)$ (line 1). Similarly, finCon$[O]$ denotes the number of the completed unit-size parts of the jobs processed on the processors of the organization O. The algorithm updates the utilities and the estimates of the contributions. The waiting jobs are assigned to the processors in the order of decreasing differences $(\phi - \psi)$ of the issuing organizations (similarly to Ref).

RoundRobin. The algorithm cycles through the list of organizations to determine the job to be started.

FairShare [10]. This is perhaps the most popular scheduling algorithm using the idea of distributive fairness. Each organization is given a target weight (a *share*). The algorithm tries to ensure that the resources used by different organizations are proportional to their shares. More formally, whenever there is a free processor and some jobs waiting for execution, the algorithm sorts the organizations in the ascending order of the ratios: the total time of the processor already assigned for the jobs of the organization divided by its share. A job from the organization with the lowest ratio is started.

In all versions of fair share, in the experiments we set the target share to the fraction of processors contributed by an organization to the global pool.

UtFairShare. This algorithm uses the same idea as FairShare. The only difference is that UtFairShare tries to balance the utilities of the organizations instead of their resource allocation. Thus, in each step the job of the organization with the smallest ratio of utility to share is selected.

CurrFairShare. This version of the fair share algorithm does not keep any history; it only ensures that, for each organization, the number of currently executing jobs is proportional to its target share.

5 Simulation Experiments

5.1 Settings

To run simulations, we chose the following workloads from the Parallel Workload Archive [5]: 1. LPC-EGEE[1] (cleaned version), 2. PIK-IPLEX[2], 3. RICC[3],

[1] http://www.cs.huji.ac.il/labs/parallel/workload/l_lpc/index.html
[2] http://www.cs.huji.ac.il/labs/parallel/workload/l_pik_iplex/index.html
[3] http://www.cs.huji.ac.il/labs/parallel/workload/l_ricc/index.html

4. SHARCNET-Whale[4]. We selected traces that closely resemble sequential workloads (in the selected traces most of the jobs require a single processor). We replaced parallel jobs that required $q > 1$ processors with q copies of a sequential job having the same duration.

In each workload, each job has a user identifier (in the workloads there are respectively 56, 225, 176 and 154 distinct user identifiers). To distribute the jobs between the organizations we uniformly distributed the user identifiers between the organizations; the job sent by the given user was assigned to the corresponding organization.

Because Ref is exponential, the experiments are computationally-intensive; in most of the experiments, we simulate only 5 organizations.

The users usually send their jobs in consecutive blocks. We also considered a scenario when the jobs are uniformly distributed between organizations (corresponding to a case when the number of users within organizations is large, in which case the distribution of the jobs should be close to uniform). These experiments led to the same conclusions, so we present only the results from the case when the user identifiers were distributed between the organizations.

For each workload, the total number of the processors in the system was equal to the number originally used in the workload (that is 70, 2560, 8192 and 3072, respectively). The processors were assigned to organizations so that the counts follow Zipf and (in different runs) uniform distributions.

For each algorithm, we compared the vector of the utilities (the utilities per organization) at the end of the simulated time period (a fixed time t_{end}): $\boldsymbol{\psi}$ with the vector of the utilities in the ideally fair schedule $\boldsymbol{\psi}^*$ (computed by Ref). Let p_{tot} denote the total number of the unit-size parts of the jobs completed in the fair schedule returned by Ref, $p_{tot} = \sum_{(s,p) \in \sigma^* : s \leq t_{end}} \min(p, t_{end} - s)$. We calculated the difference $\Delta\psi = |\boldsymbol{\psi} - \boldsymbol{\psi}^*| = \sum_{O^{(u)}} (\psi^{(u)} - \psi^{(u),*})$ and compared the values $\Delta\psi / p_{tot}$ for different algorithms. The value $\Delta\psi / p_{tot}$ is the measure of the fairness that has an intuitive interpretation. Since delaying each unit-size part of a job by one time moment decreases the utility of the job owner by one, the value $\Delta\psi / p_{tot}$ gives the average unjustified delay (or unjustified speed-up) of a job due to the unfairness of the algorithm.

5.2 Results

We start with experiments on short sub-traces of the original workloads. We randomly selected the start time of the experiment t_{start} and set the end time to $t_{end} = t_{start} + 5 \cdot 10^4$. For each workload we run 100 experiments (on different periods of workloads of length $5 \cdot 10^4$). The average values of $\Delta\psi / p_{tot}$, and the standard deviations are presented in Table 1.

From this part of the experiments we conclude that: (i) The algorithm Rand is the most fair algorithm regarding the fairness by the Shapley Value; but Rand is the second most computationally intensive algorithm (after Ref). (ii) All the other algorithms are about equally computationally efficient.

[4] http://www.cs.huji.ac.il/labs/parallel/workload/l_sharcnet/index.html

Table 1. The average delay (or the speed up) of jobs due to the unfairness of the algorithm $\Delta\psi/p_{tot}$ for different algorithms and different workloads. Each row is an average over 100 instances taken as parts of the original workload. The duration of the experiment is $5 \cdot 10^4$

	LPC-EGEE		PIK-IPLEX		SHARCNET-Whale		RICC	
	Avg	St. dev.	Avg	St. dev.	Avg	St. dev.	Avg	St. dev.
RoundRobin	238	353	6	33	145	38	2839	357
Rand ($N = 15$)	8	21	0.014	0.01	6	6	162	187
DirectContr	5	11	0.02	0.15	10	7	537	303
FairShare	16	25	0.3	1.38	13	8	626	309
UtFairShare	16	25	0.3	1.38	38	67	515	284
CurrFairShare	87	106	0.3	1.58	145	80	1231	243

Table 2. The average delay (or the speed up) of jobs due to the unfairness of the algorithm $\Delta\psi/p_{tot}$ for different algorithms and different workloads. Each row is an average over 100 instances taken as parts of the original workload. The duration of the experiment is $5 \cdot 10^5$

	LPC-EGEE		PIK-IPLEX		SHARCNET-Whale		RICC	
	Avg	St. dev.	Avg	St. dev.	Avg	St. dev.	Avg	St. dev.
RoundRobin	4511	6257	242	1420	404	1221	10850	13773
Rand ($N = 15$)	562	1670	1.3	7	26	158	771	1479
DirectContr	410	1083	0.2	1.4	60	204	1808	3397
FairShare	575	1404	2.3	12	94	307	2746	4070
UtFairShare	888	2101	1.2	5	120	344	4963	6080
CurrFairShare	1082	2091	2.2	11	180	805	5387	9083

The algorithm DirectContr is the most fair. (iii) The algorithm FairShare, which is the algorithm mostly used in real systems, is not much worse than DirectContr. (iv) Arbitrary scheduling algorithms like RoundRobin may result in unfair schedules. (v) The fairness of the algorithms may depend on the workload. In RICC the differences are much more visible than in PIK-IPLEX. Thus, although DirectContr and FairShare are usually comparable, on some workloads the difference is important.

In the second series of experiments, we verified the effect of the duration of the simulated workload on the resulting fairness measure (the ratio $\Delta\psi/p_{tot}$). As we changed the duration of the experiments from $5 \cdot 10^4$ to $5 \cdot 10^5$, we observed that the unfairness ratio $\Delta\psi/p_{tot}$ was increasing. The value of the ratio for $t_{end} - t_{start} = 5 \cdot 10^5$ are presented in Table 2. The relative quality of the algorithms is the same as in the previous case. Thus, all our previous conclusions hold. However, now all the algorithms are significantly less fair than the exact algorithm. Thus, in long-running systems the difference between the approaches becomes more important. If there are a few organizations, the exact Ref or the

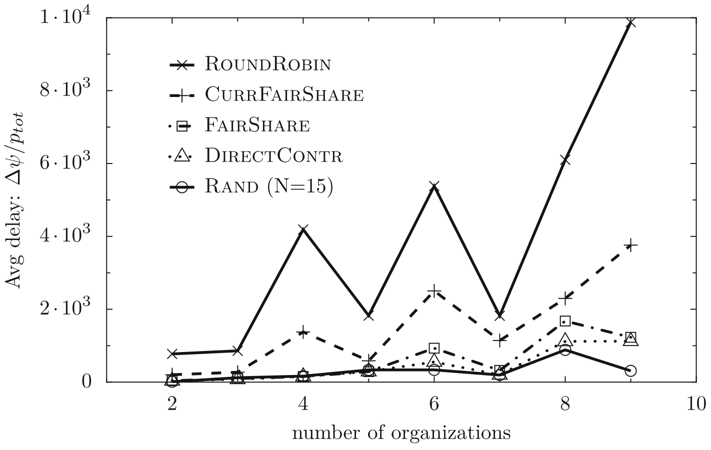


Fig. 1. The effect of the number of the organizations on ratio $\Delta\psi/p_{tot}$

randomized Rand algorithms should be used. In larger systems, when the computational cost of these is too high, DirectContr clearly outperforms FairShare.

Last, we verified the effect of the number of the organizations on the ratio $\Delta\psi/p_{tot}$. The results from the experiments conducted on LPC-EGEE data set are presented in Fig. 1. As the number of organizations increases, the unfairness ratio $\Delta\psi/p_{tot}$ grows; thus the difference between the algorithms is more significant. This confirms our previous conclusions. The relative fairness of the algorithms is the same as in our previous experiments.

6 Conclusions

In this paper we present DirectContr, a heuristic algorithm for the problem of Shapley-fair scheduling. We conduct extensive experimental evaluation of the fairness of our algorithm comparing it to other algorithms used in real systems. We conclude that the randomized algorithm is the closest to the referral exponential algorithm, yet it is also the most computationally intensive. Among computationally-tractable algorithms, DirectContr, our heuristic, is the closest to the referral algorithm, although on shorter workloads with relatively few organizations, the fair share algorithm is similar. The difference between algorithms becomes significant in longer-running systems with many organizations. The main conclusion from our work is that in multi-organizational systems, the distributive fairness used by fair share does not result in truly-fair schedules; our heuristic better approximates the Shapley-fair schedules.

Acknowledgements. This work is partly supported by Polish National Science Center Sonata grant UMO-2012/07/D/ST6/02440

References

1. Carroll, T.E., Grosu, D.: Divisible load scheduling: an approach using coalitional games. In: Proceedings of the ISPDC (2007)
2. Chaskar, H.M., Madhow, U.: Fair scheduling with tunable latency: a round-robin approach. IEEE/ACM Trans. Netw. **11**(4), 592–601 (2003)
3. Cohen, J., Cordeiro, D., Trystram, D., Wagner, F.: Multi-organization scheduling approximation algorithms. Concurr. Comput. Pract. Exp. **23**(17), 2220–2234 (2011)
4. Dutot, P.-F., Pascual, F., Rzadca, K., Trystram, D.: Approximation algorithms for the multi-organization scheduling problem. IEEE Trans. Parallel Distrib. Syst. **22**, 1888–1895 (2011)
5. Feitelson, D.G.: Parallel workloads archive. http://www.cs.huji.ac.il/labs/parallel/workload/
6. Goyal, P., Vin, H.M., Chen, H.: Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. In: Proceedings of the SIGCOMM, pp. 157–168 (1996)
7. Gulati, A., Ahmad, I.: Towards distributed storage resource management using flow control. SIGOPS Oper. Syst. Rev. **42**(6), 10–16 (2008)
8. Gulati, A., Ahmad, I., Waldspurger, C.A.: PARDA: proportional allocation of resources for distributed storage access. In: FAST Proceedings, February 2009
9. Jin, W., Chase, J.S., Kaur, J.: Interposed proportional sharing for a storage service utility. SIGMETRICS Perform. Eval. Rev. **32**(1), 37–48 (2004)
10. Kay, J., Lauder, P.: A fair share scheduler. Commun. ACM **31**(1), 44–55 (1988)
11. Kostreva, M.M., Ogryczak, W., Wierzbicki, A.: Equitable aggregations and multiple criteria analysis. Eur. J. Oper. Res. **158**(2), 362–377 (2004)
12. Mashayekhy, L., Grosu, D.: A merge-and-split mechanism for dynamic virtual organization formation in grids. In: PCCC Proceedings, pp. 1–8 (2011)
13. Mishra, D., Rangarajan, B.: Cost sharing in a job scheduling problem using the Shapley value. In: Proceedings of the EC, pp. 232–239 (2005)
14. Moulin, H.: On scheduling fees to prevent merging, splitting, and transferring of jobs. Math. Oper. Res. **32**(2), 266–283 (2007)
15. Osborne, M.J., Rubinstein, A.: A Course in Game Theory, vol. 1. MIT Press, Cambridge (1994)
16. Rzadca, K., Trystram, D., Wierzbicki, A.: Fair game-theoretic resource management in dedicated grids. In: CCGRID Proceedings (2007)
17. Skowron, P., Rzadca, K.: Non-monetary fair scheduling – cooperative game theory approach. In: SPAA (see also the extended arxiv version) (2013)
18. Wang, Y., Merchant, A.: Proportional-share scheduling for distributed storage systems. In: FAST Proceedings, pp. 4–4 (2007)

Setting up Clusters of Computing Units to Process Several Data Streams Efficiently

Daniel Millot and Christian Parrot(✉)

Telecom SudParis, Institut Mines-Telecom, Évry, France
{Daniel.Millot,Christian.Parrot}@mines-telecom.fr

Abstract. Let us consider an upper bounded number of data streams to be processed by a Divisible Load application. The total workload is unknown and the available speeds for communicating and computing may be poorly a priori estimated. This paper presents a resource selection method that aims at maximizing the throughput of this processing. From a set of processing units linked by a network, this method consists in forming an optimal set of master-workers clusters. Results of simulations are presented to assess the efficiency of this method experimentally. Before focusing on the proposed resource selection method, the paper comes back on the adaptive scheduling method on which it relies.

Keywords: Adaptive scheduling · Parallel processing · Master-worker model · Load balancing · Heterogeneous context · Dynamic context

1 Introduction and Related Works

We consider an application that processes data acquired as streams, during an a priori unknown time-lapse. The throughput of the streams is supposed to be unlimited. Each data stream is arbitrarily associated to one processing unit in order to be processed. We assume that each data stream can be split into chunks as small as necessary for the scheduling and that each chunk can be processed independently of the others. Such applications, that can be found in many different domains [1,2], look like a Divisible Load application [3], but we assume that the total workload is unknown.

Each processing unit which acquires a data stream can split the flow into chunks and distribute the chunks to be processed to other computation units, via a network. Each unit which receives a load can process it and send back the result of its processing; as a lot of data parallel applications [4,5]. We assume that the communication speeds are high enough, in relation to the computation speeds, make the execution of this application to benefit from an execution in parallel.

According to the master-workers paradigm, the processing units which distribute the load act as masters and the computation units which do the processing act as workers [6]. Besides, we assume that the computation units have an unlimited buffering capability, unlike [7]. The resources, for both communicating and processing, can be heterogeneous as for [8,9] and the durations of communicating and processing a chunk are supposed to be affine with respect to the chunk size.

Let us assume that the available communication speeds, available computation speeds and latencies that characterize these resources, and that we call execution parameters in the sequel, may be inaccurately specified; only estimates of the execution parameters are a priori available to the scheduler.

We assume that computation can overlap communication. We consider a 1-port bi-directional communication model between master and workers. This model allows a communication from master to worker to overlap a communication from worker to master. But, with such a communication model, only one communication from, or to, one processing unit (master or worker) can be performed at the same time. This feature fits in with the behavior of message passing communications when messages are big enough [10].

We call "round" a sequence of consecutive actions leading the master to feed all the workers once and collect the corresponding results. As a result of the unawareness of the total workload, the minimization of the makespan is impossible and the scheduling must proceed in an iterative way, as the workload flows in. Therefore the scheduling has to be multi-round.

The processing of each data stream can be split into three phases. The start-up phase begins when the master starts to send the very first chunk to a worker and ends when each worker has received a chunk to process. Then begins the streaming phase which ends when the last data item in the stream has been acquired by the master. Eventually, the cleanup phase begins and it lasts until the master has received the very last result of processing of each worker.

When the cleanup phase begins, the total workload happens to be known and the makespan can be minimized by making the workers complete their work simultaneously [11]. In order to reduce as much as possible the cost of communication between workers to reallocate the chunks so as to balance their remaining load, when the cleanup phase begins, it is convenient to upper-bound the workload discrepancy between workers during the streaming phase. A usual way to do that is to make asymptotically periodic the distribution of the chunks, during the streaming phase. This strategy has the extra-advantage of facilitating the reduction of the risk of contentions when workers compete to communicate with the master [12].

Later on in this paper, we will consider that the start-up duration is negligible compared to the streaming phase duration; unlike [13,14]. The legitimacy of this assumption increases with the ratio of the total workload to the load distributed at the very first round. Under this assumption it is reasonable to focus on the streaming phase and to aim at maximizing the throughput during this period.

In order to iteratively adjust the schedule to the possible inaccuracy of the specification of the execution parameter (and possibly even to their variation over time), it is relevant to arbitrarily set the frequency of the successive steps of this adaptation. The higher this frequency, in other words, the smaller the chunk sizes, the faster the adaptation; but unfortunately the longer the time wasted in latencies. We want the adjustment of the schedule for the next round to be based only on the measurement of durations like those of communicating or processing of the previous rounds. Indeed, this measurement is the best benchmark to estimate the actual value of the execution parameters.

Based on these remarks, the AS4DR (Adaptive Scheduling for Distributed Resources) method [15] automatically adapts the schedule to both: the heterogeneity of the workers and the poorness of the estimation of the execution parameters. But, it deals with only one data stream, processed by only one master-workers platform. This method sets the chunk size for each worker at each round by assuming that the execution parameters for the next round will be identical to the one of the previous round. Let $\alpha_{\mathrm{w,i}}$ be the size of the chunk sent to a worker w for round i. Let τ and $\sigma_{\mathrm{w,i}}$ be respectively the wanted and the measured time durations between the start of the sending of a chunk of size $\alpha_{\mathrm{w,i}}$ and the end of the reception of the corresponding result by the master. The basic idea of the AS4DR multi-round method is to adapt $\alpha_{\mathrm{w,i}}$ according to:

$$\alpha_{\mathrm{w,i}} := \alpha_{\mathrm{w,i-1}} \frac{\tau}{\sigma_{\mathrm{w,i-1}}} \qquad \text{for i} > 1. \tag{1}$$

It has been proved [12] that with a multi-port communication model and linear costs for computations and communications, an asymptotic periodic schedule can be installed without knowing the execution parameters. This result can cope with an heterogeneous but steady-state context. Unfortunately this approach generates idleness of the workers between the processing of successive chunks, while the workers communicate with their master. Taking over this principle, AS4DR prevents this kind of idleness by making computation overlap communication. The asymptotic periodicity of the schedule with AS4DR remains when the costs for computations and communications are affine. With the aid of the estimate of the execution parameters and, before the launch of the AS4DR scheduler, a step called CIP (for Contentions and Idleness Prevention) determines a value for τ, large enough to involve without contention all the workers into the scheduling [15]. It has been proved that AS4DR maximizes the CPU-efficiency of the workers in a heterogeneous but steady-state context [16]. Tests that have been performed experimentally assess these theoretical results, when considering a single master-workers platform. More, other tests have shown that AS4DR can adapt the schedule to not only the poorness of the estimates, but to the variation over time of the execution parameters as well [17]. But using an evermore increasing amount of resources makes the increase of the value of τ ultimately impossible, without reconsidering the hypothesis according to which the workload for a round is negligible with respect to the total workload. Section 2 reminds the way AS4DR schedules when considering only one data stream and only one master-workers platform.

To overcome this scaling-up difficulty, the constraint which bounds the number of data stream to one can be relaxed. Let M be an upper bound of the number of data streams. In this new context the preliminary step CIP is replaced by another one which, for a given value of τ, selects workers to form master-workers clusters so that contentions are avoided. So, AS4DR is able to install an asymptotic periodic schedule for each data stream which, in the absence of contentions thus obtained, prevents the idleness of the workers; with the exception of latencies. The choice of the value of τ depends on several criteria. Of course, to reduce as much as possible the time wasted in latencies, the value of τ should be as large as possible. But, on the other hand, the smaller τ, the faster the adaptation of the schedule to the poorness of the estimate of the execution parameters and also the more balanced the remaining workload between workers when the cleanup phase begins. When the execution parameters vary over time, the value of τ should be smaller than the smallest steady state period. When the upper bound M of the number of data streams is big enough with respect to the number of available workers, all workers can be involved into the processing. In this case, as all workers are fully used, the global throughput is maximum ; for a given value of τ. On the contrary, if some workers are not selected (because of their lesser profitability for the global throughput), the throughput is not necessarily maximum, but the selected workers are fully used. Section 3 presents this preliminary step. Finally, we conclude in Sect. 4.

2 Presentation of the AS4DR Method

To allow the overlapping between communication and computation, the AS4DR scheduler splits each chunk it has to deliver to a worker into two subchunks. So sending subchunks of arbitrarily chosen sizes $\dot{\alpha}_{\mathrm{w},1}$ and $\ddot{\alpha}_{\mathrm{w},1}$ to each worker w for the first round, the AS4DR scheduler then sends to worker w, for each round i, two subchunks $\dot{\mathbf{s}}$ and $\ddot{\mathbf{s}}$ of respective sizes $\dot{\alpha}_{\mathrm{w,i}}$ and $\ddot{\alpha}_{\mathrm{w,i}}$ such that: $\dot{\alpha}_{\mathrm{w,i}} + \ddot{\alpha}_{\mathrm{w,i}} = \alpha_{\mathrm{w,i}}$.

Let θ_{w} denotes the constant ratio between $\dot{\alpha}_{\mathrm{w,i}}$ and $\alpha_{\mathrm{w,i}}$. From estimates of the execution parameters, a preliminary step (for instance CIP) sets τ and $\underset{0\leq \mathrm{w}\leq \mathrm{N}-1}{(\alpha_{\mathrm{w},1}, \theta_{\mathrm{w}})}$; where N denotes the number of workers. The round i for worker w is composed of three phases: transmission in a row of the data from master to worker for subchunks $\dot{\mathbf{s}}$ and $\ddot{\mathbf{s}}$, worker computation on the received data, transmission in a row of the computation result from worker to master for subchunks $\dot{\mathbf{s}}$ and $\ddot{\mathbf{s}}$. It is worth noticing that the result corresponding to the $\ddot{\mathbf{s}}$ subchunk of some round is returned to the master just after the result corresponding to the $\dot{\mathbf{s}}$ subchunk of the next round has itself been returned. Let us denote $\dot{\mathrm{C}}_{\mathrm{w,i}}$ the measured time spent to process the subchunk $\dot{\mathbf{s}}$ of round i and f_{w} the latency of computation, for worker w. We define

$$\sigma_{\mathrm{w,i}} \equiv \frac{\dot{\mathrm{C}}_{\mathrm{w,i}} - \mathrm{f}_{\mathrm{w}}}{\theta_{\mathrm{w}}} + 2\mathrm{f}_{\mathrm{w}}. \tag{2}$$

When the communications between master and workers are contention-free, and if the workload is set with the aid of (1) and (2), then the effective duration of the rounds linearly converges to τ for any worker; whatever the initial (strictly

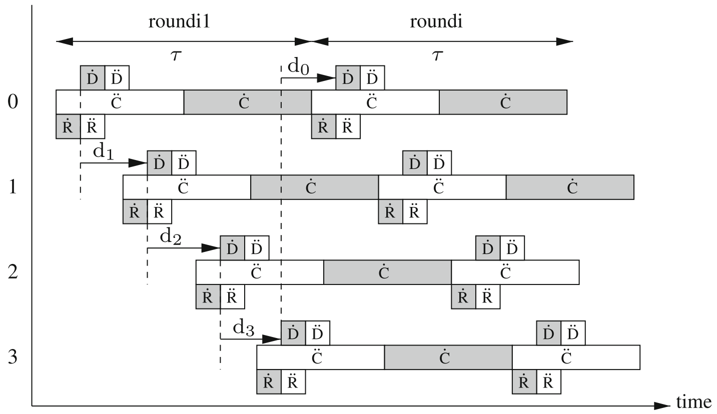


Fig. 1. Contention-free asymptotic schedule

positive) workload. So, the schedule established according to the AS4DR method is stable within the meaning of Lyapounov and, in addition, is asymptotically stable [16].

Besides, there exists [16] a lower bound and an upper bound: $\theta^{\min}_{w,i}$ and $\theta^{\max}_{w,i}$ such that, in the absence of contentions, AS4DR method prevents the idleness between the processing of two successive subchunks, if and only if, $\theta^{\max}_{w,i+1} \geq \theta_w \geq \theta^{\min}_{w,i+1}$, $\forall$i.

To prevent contentions, the instant each worker accesses to the master is set far enough from the instants the others access too, by introducing time delays d_w before posting the very first subchunk to each worker w. Figure 1 illustrates the model of asymptotic τ-periodic (thus round-robin) schedule we are seeking, when N=4. Let us define the delay d_w:

$$d_w \equiv (1+\lambda_w) \max\left(\overline{\dot{D}_{w-1}} + \overline{\ddot{D}_{w-1}}, \overline{\ddot{R}_{w-1}} + \overline{\dot{R}_w}\right) \text{ (modulo N)}; \tag{3}$$

where: λ_w stands for a positive constant factor (the greater the inaccuracy of the estimate of the execution parameters, the greater λ_w), $\overline{\dot{D}_w}$ (resp. $\overline{\ddot{D}_w}$, $\overline{\dot{R}_w}$ and $\overline{\ddot{R}_w}$) denotes an a priori estimate of the time spent to communicate the $\dot{\mathbf{s}}$ data (resp. $\ddot{\mathbf{s}}$ data, $\dot{\mathbf{s}}$ result and $\ddot{\mathbf{s}}$ result), for worker w. These durations can be computed with the help of a priori estimates of the execution parameters [16].

Besides, the time intervals d_w should allow all the workers to be served during the first round, i.e. within a τ period. Thus, to prevent contention, it is sufficient that τ verify:

$$\sum_{w=0}^{N-1} d_w \leq \tau. \tag{4}$$

3 Resource Selection for AS4DR in a Multiple Data Streams Context

3.1 Method

Let $\mathcal{W}$ be the set of all available workers. Let $\mathrm{x_{m,w}}$ equals one if the worker w is selected in the cluster m, and $\mathrm{x_{m,w}}$ equals zero otherwise. As one worker can belong to one cluster at most,

$$\sum_{\mathrm{m}=0}^{\mathrm{M}-1} \mathrm{x_{m,w}} \leq 1, \forall \mathrm{w} \in \mathcal{W}. \tag{5}$$

Let T be the global throughput,

$$\mathrm{T} = \sum_{\mathrm{m}=0}^{\mathrm{M}-1} \sum_{\mathrm{w} \in \mathcal{W}} \mathrm{t_w x_{m,w}}; \tag{6}$$

where $\mathrm{t_w}$, the potential throughput of worker w in the absence of idleness between the processing of any successive subchunks, verifies:

$$\mathrm{t_w} = \left(1 - 2\frac{\mathrm{f_w}}{\tau}\right) \mathrm{F_w};$$

where $\mathrm{F_w}$ and $\mathrm{f_w}$ respectively denote the available computation speed and the computation latency, of worker w. In order to prevent contentions, and thus to make the use of AS4DR possible, we have seen (4) that necessarily,

$$\sum_{\mathrm{w} \in \mathcal{W}} \mathrm{d_w x_{m,w}} \leq \tau_{\mathrm{m}}, \quad \forall\ 0 \leq \mathrm{m} \leq \mathrm{M} - 1; \tag{7}$$

where τ_{m} is the wanted duration for the rounds of cluster m. To form a set of clusters which maximizes the global throughput, when considering M data streams at most, we want to find $(\mathrm{x_{m,w}})_{0 \leq \mathrm{m} \leq \mathrm{M}-1, \mathrm{w} \in \mathcal{W}}$ that maximizes T, given by (6), subject to constraints (7) and (5). This problem looks like a Multiple Knapsack problem. But as the value of τ_{m} cannot be set a priori, since it depends on the workers selected for cluster m, the same wanted duration τ is set for the rounds of all clusters. This choice has the advantage of upper bounding the load discrepancy between the different clusters uniformly.

Besides, the time lag $\mathrm{d_w}$, defined by (3), depends on the worker $\mathrm{w} - 1$, the one that precedes w in the round robin distribution for the cluster which w belongs to. As this worker $\mathrm{w} - 1$ is a priori unknown, we need to reformulate the Multiple Knapsack problem to make the weight for worker w be independent from worker $\mathrm{w} - 1$. For that, let us tighten up the constraints of the problem. We define $\mathring{\mathrm{d}}_{\mathrm{w}}$, as follows:

$$\mathring{\mathrm{d}}_{\mathrm{w}} \equiv (1 + \lambda_{\mathrm{w}}) \max\left(\overline{\mathrm{D_{max}}}, \overline{\ddot{\mathrm{R}}_{\mathrm{max}}} + \overline{\dot{\mathrm{R}}_{\mathrm{w}}}\right); \text{ where } \begin{cases} \overline{\mathrm{D_{max}}} \equiv \max\limits_{\mathrm{w} \in \mathcal{W}} \left(\overline{\dot{\mathrm{D}}_{\mathrm{w}}} + \overline{\ddot{\mathrm{D}}_{\mathrm{w}}}\right), \\ \overline{\ddot{\mathrm{R}}_{\mathrm{max}}} \equiv \max\limits_{\mathrm{w} \in \mathcal{W}} \overline{\ddot{\mathrm{R}}_{\mathrm{w}}}. \end{cases}$$

As $\mathrm{d_w}$ is smaller than $\overset{\circ}{\mathrm{d}}_\mathrm{w}$, the feasible space defined by the following constraints:

$$\sum_{\mathrm{w}\in\mathcal{W}} \overset{\circ}{\mathrm{d}}_\mathrm{w}\mathrm{x_{m,w}} \leq \tau, \quad \forall\ 0 \leq \mathrm{m} \leq \mathrm{M}-1, \tag{8}$$

is a subspace of the one previously defined by (7). So, the tightened problem we have to solve now is: find $(\mathrm{x_{m,w}})_{0\leq \mathrm{m}\leq \mathrm{M}-1,\mathrm{w}\in\mathcal{W}}$ that maximizes T, given by (6), subject to constraints (8) and (5).

Unfortunately, the coefficients $\mathrm{t_w}$, $\overset{\circ}{\mathrm{d}}_\mathrm{w}$ and τ are not positive integers, like for the classical Multiple Knapsack problem. To overcome this last complication, the resource selection problem is reformulated. According to the machine number representation, $\mathrm{t_w}$, $\overset{\circ}{\mathrm{d}}_\mathrm{w}$ and τ are rational numbers. So, in order to make the problem be formulated like a Multiple Knapsack problem, (6) can be multiplied by the least common multiple of the denominators of $(\mathrm{t_w})_{w\in\mathcal{W}}$ whereas, (8) can be multiplied by the least common multiple of the denominators of $(\mathrm{d_w})_{w\in\mathcal{W}}$ and τ. This new problem can be solved in a classical way by a Multiple Knapsack method. If the whole set of workers happens to be selected to participate in the processing, then the solution of this problem maximizes the global throughput; for a given wanted duration for the rounds τ. Of course, as the presented method to select the workers is based on the underlying scheduling method: AS4DR, nothing allows one to claim that the global throughput is optimal if only a part of the whole set of workers happens to be selected. Of course, there exists a value of M big enough to make all the workers be ultimately involved into the schedule. Because of the successive transformations of the resource selection problem, the number of actually formed clusters is possibly no more minimal (but still smaller than M).

The next section is devoted to the experimental assessment of the resource selection method. The method described in [18] has been chosen to solve the Multiple Knapsack problem posed by the resource selection.

3.2 Experimental Assessment

All the simulations, which results are presented in this section, have been conducted with the SimGrid framework [19]. We consider 10 sets $(\mathrm{S_k})_{0\leq \mathrm{k}\leq 9}$ of values for the execution parameters of the workers, given (speeds in bytes/second and latencies in seconds) in Table 1. Each of these sets is randomly a priori allocated to 100 workers among the 1000 available workers. Let us denote, $\mathrm{B_w^D}$ (resp. $\mathrm{B_w^R}$) the available communication speed of the link from the master to worker w (resp. from worker w to the master). To make $\mathrm{F_w}$, $\mathrm{B_w^D}$ and $\mathrm{B_w^R}$ varying over time, in a way that facilitates the correlation of the variations with their effects on the scheduling, each of these parameters can only take two values. According to the 10 profiles $(\mathrm{P_k})_{0\leq \mathrm{k}\leq 9}$ of variation shown in Fig. 2, $\left(\mathrm{F_w},\ \mathrm{B_w^D},\ \mathrm{B_w^R}\right)$ alternatively takes the reference value (in Table 1): $\left((\mathrm{F_w})_\mathrm{ref},\ \left(\mathrm{B_w^D}\right)_\mathrm{ref},\ \left(\mathrm{B_w^R}\right)_\mathrm{ref}\right)$, and the perturbed value: $\left((1-\delta_\mathrm{k})\,(\mathrm{F_w})_\mathrm{ref},\ (1-\delta_\mathrm{k})\left(\mathrm{B_w^D}\right)_\mathrm{ref},\ (1-\delta_\mathrm{k})\left(\mathrm{B_w^R}\right)_\mathrm{ref}\right)$; where δ_k takes a strictly positive value given in Table 2. Each profile is randomly

Table 1. Reference values

	Computation		Communication master $\longrightarrow$ w_i		Communication master $\longleftarrow$ w_i		Number of workers
	Speed	Latency	Speed	Latency	Speed	Latency	
S_0	1.0e+0	1.0e−2	1.0e+6	1.0e−2	1.0e+6	1.0e−2	100
S_1	1.0e+2	1.0e−1	1.0e+7	1.0e−3	1.0e+7	1.0e−3	100
S_2	1.0e+1	1.0e−3	1.0e+8	1.5e−2	1.0e+8	1.5e−2	100
S_3	1.0e+3	1.0e−2	1.0e+9	1.0e−2	1.0e+9	1.0e−2	100
S_4	1.0e+0	1.0e−3	1.0e+6	1.0e−2	1.0e+6	1.0e−2	100
S_5	1.0e+2	1.0e−1	1.0e+7	1.0e−3	1.0e+7	1.0e−3	100
S_6	1.0e+1	1.0e−4	1.0e+8	1.0e−2	1.0e+8	1.0e−2	100
S_7	1.0e+3	1.0e−2	1.0e+9	1.0e−2	1.0e+9	1.0e−2	100
S_8	1.0e+0	1.0e−3	1.0e+6	1.0e−3	1.0e+6	1.0e−3	100
S_9	1.0e+2	1.0e−2	1.0e+7	1.0e−2	1.0e+7	1.0e−2	100

Table 2. Dynamic values

δ_0	0.0
δ_1	0.1
δ_2	0.2
δ_3	0.3
δ_4	0.4
δ_5	0.5
δ_6	0.6
δ_7	0.7
δ_8	0.8
δ_9	0.9

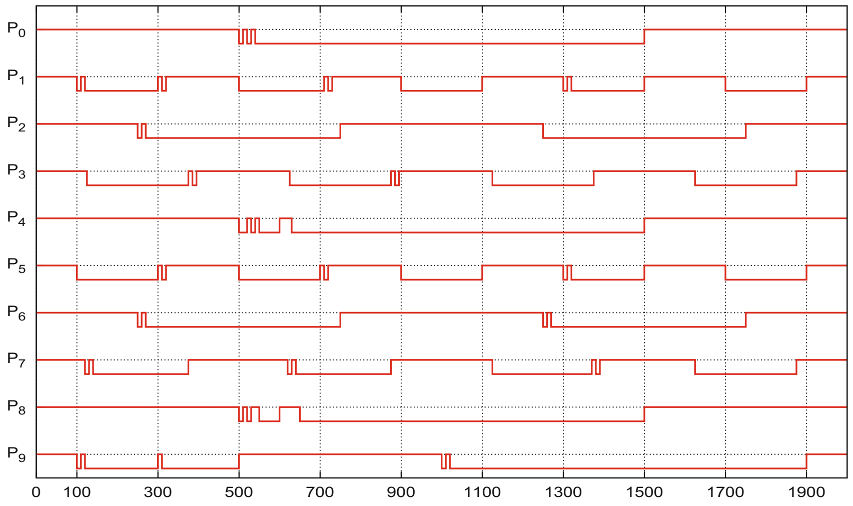


Fig. 2. Perturbed execution parameter as a function of time (in seconds)

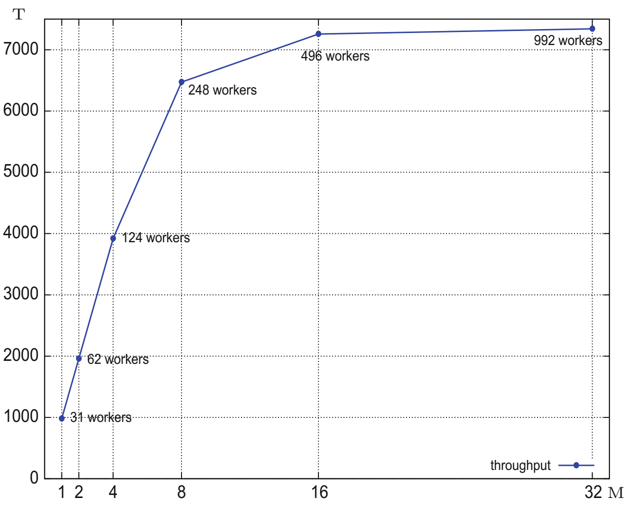


Fig. 3. Throughput as a function of the upper bound of the number of data streams: M

allocated to 100 workers. In this context, the coefficient δ_k characterizes the variations of the execution parameters and is called "dynamicity" in the sequel; with δ_9 the amplitude of the variation of the execution parameters is maximum, whereas it is minimum (steady state context) with δ_0. For each simulation, the dynamicity is the same for all the workers.

Figure 3 shows the variation of the global throughput (in bytes/second) as a function of M, the upper bound of the number of data streams. For each value of M, the total number of selected workers is given. The execution parameters for this simulation are those of Table 1; dynamicity equals δ_0, and the a priori estimate of the execution parameters equals their real values. As best workers are selected in priority, the greater the number of workers involved in the scheduling, the lower the increase of the throughput.

Figure 4 represents the mean value of the measured durations (in seconds) of the very first rounds: σ, for all selected workers, as a function of the elapsed time (in seconds), when τ equals 100 s, in steady state context (dynamicity=δ_0). For this experiment, the initial workload is set to a value that could have resulted from a computation based on poor estimates of the execution parameters. This initial workload is set to $-80\,\%$ of the load computed with the aid of the real execution parameters; those of Table 1, when the latency for computation, for all the workers, is set to: 0.1, 0.5 and 0.9, successively. Figure 4 shows the convergence towards τ of the mean value of σ. As expected, the smaller the processing latencies, the faster the adaptation of σ to τ. Figure 4 illustrates the adaptation of the workload to the poorness of the estimation of the execution parameters.

Although the throughput remains the ultimate performance indicator, the throughput does not highlight the rate of time really spent in processing.

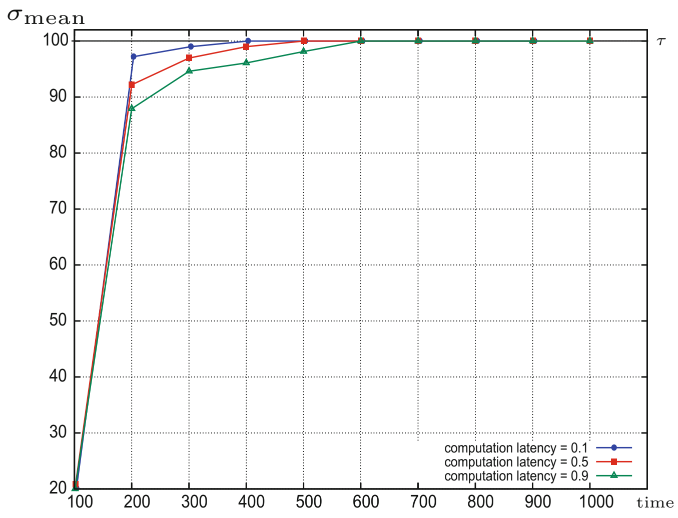


Fig. 4. Mean value of s as a function of elapsed time

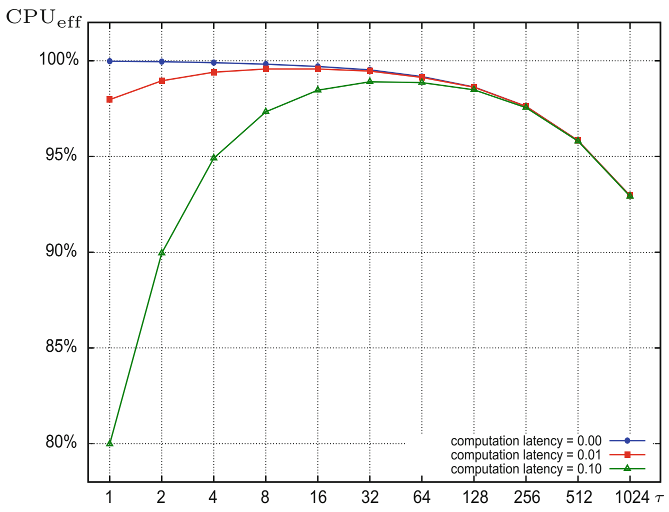


Fig. 5. CPU_{eff} as a function of τ, for several values of the dynamicity $= \delta_0$

Thus, let us define CPU_{eff} the CPU-efficiency:

$$\text{CPU}_{\text{eff}} \equiv 1 - \frac{\text{CPU idleness}}{\text{elapsed time}}.$$

Figure 5 illustrates that the way to set the value of τ must depend on the computation latency; as long as these latencies are not negligible compared to τ. This simulation has been performed in steady state context (dynamicity=δ_0),

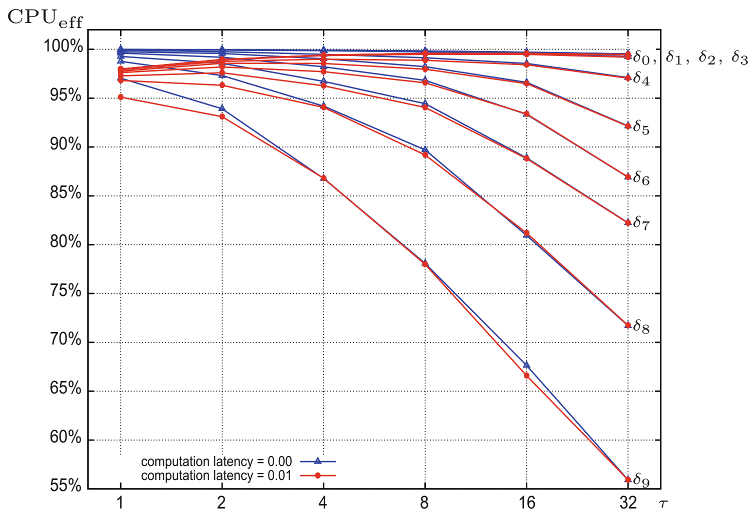


Fig. 6. CPU$_{\text{eff}}$ as a function of τ, for several values of the dynamicity

with the full knowledge of the execution parameters of Table 1, but the computation latencies for all the workers are replaced by the values: 0.0, 0.01 and 0.1, successively. Figure 5 also reminds one that, according to the value of the computation latency, there exists an optimal value for τ. The existence of such an optimal value is due to a compromise between two necessities. It is necessary to make τ as great as possible in order to minimize the time wasted in latencies; by reducing the number of rounds. On the other hand, the smaller the value of τ, the earlier the worker starts to process.

In Fig. 6 we can see that the higher the dynamicity, the smaller τ must be. Figure 6 also confirms that, even in a dynamic context, the smaller τ is, the higher the effects on the efficiency of the time wasted in latencies are. This simulation has been performed with several values of dynamicity, with the execution parameters of Table 1, but the CPU latencies for all the workers were replaced by the values: 0.0 and 0.01, successively.

4 Conclusion

This paper addresses the problem of setting up clusters of heterogeneous distributed resources to process several data streams of a Divisible Load application. As the total workload is unknown, a very first workload, corresponding to the wanted duration for the very first round: τ, must be a priori set from estimates of the characteristics of the resources. To facilitate the contention prevention, the wanted duration for all the rounds and for all workers has been arbitrarily set to τ. For the moment the value of τ is empirically set, according to both the poorness of the a priori estimate of the execution parameters and the frequency of their possible variation over time. If only a part of the whole set of available computation units are involved into the scheduling, then we can only claim that the selected computation units are fully used. If the whole set of available processing units are involved into the scheduling, then the presented method succeeds in maximizing the global throughput, for an a priori set value of τ. Compared to using pure hardware performance figures, such as bandwidth or CPU frequency to adapt the workload at each round, the method has the extra advantage of taking into account characteristics of the software such as algorithmic complexity.

The way to adapt the workload at each round generates a risk of instability, when the execution parameters vary over time. A study of the stability of the method should lead to tighten up the way τ is set and should help to design new patterns of adaptation of the chunksize.

References

1. Lee, C., Hamdi, M.: Parallel image processing application in a network of workstation. Parallel Comput. **21**, 137–160 (1995)
2. Altılar, D.T., Paker, Y.: An optimal scheduling algorithm for stream based parallel video processing. In: Yazıcı, A., Şener, C. (eds.) ISCIS 2003. LNCS, vol. 2869, pp. 731–738. Springer, Heidelberg (2003)
3. Robertazzi, T.G.: Ten reasons to use divisible load theory. IEEE Comput. **36**(5), 63–68 (2003)
4. Altilar, D., Paker, Y.: An optimal scheduling algorithm for parallel video processing. In: Proceedings of the International Conference on Multimedia Computing and Systems. IEEE Computing Society Press (1998)
5. Dong, L., Bharadwaj, V., Ko, C.C.: Efficient movie retrieval strategies for movie-on-demand multimedia services on distributed networks. Multimedia Tools Appl. **20**(2), 99–133 (2003)
6. Beaumont, O., Casanova, H., Legrand, A., Robert, Y., Yang, Y.: Scheduling divisible loads on star and tree networks: results and open problems. IEEE Trans. Parallel Distrib. Syst. **16**(3), 207–218 (2005)
7. Drozdowski, M., Wolniewicz, P.: Optimizing divisible load scheduling on heterogeneous stars with limited memory. Eur. J. Oper. Res. **172**(2), 545–559 (2006)
8. Rosenberg, A.L., Chiang, R.C.: Toward understanding heterogeneity in computing. In: Proceeding of the 24th International Parallel and Distributed Processing Symposium (IPDPS'10), vol. 1, pp. 1–10. IEEE Computing Society Press, April 2010
9. Beaumont, O., Marchal, L., Robert, Y.: Scheduling divisible loads with return messages on heterogeneous master-worker platforms. In: Bader, D.A., Parashar, M., Sridhar, V., Prasanna, V.K. (eds.) HiPC 2005. LNCS, vol. 3769, pp. 498–507. Springer, Heidelberg (2005)
10. Saif, T., Parashar, M.: Understanding the behavior and performance of non-blocking communications in MPI. In: Danelutto, M., Vanneschi, M., Laforenza, D. (eds.) Euro-Par 2004. LNCS, vol. 3149, pp. 173–182. Springer, Heidelberg (2004)
11. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.: Scheduling divisible loads in parallel and distributed systems. IEEE Computing Society Press, Los Almitos (1996)

12. Drozdowski, M.: Selected problems of scheduling tasks in multiprocessor computing systems. Ph.D. thesis, Instytut Informatyki Politechnika Poznanska, Poznan (1997)
13. Bharadwaj, V., Ghose, D., Mani, V.: Multi-installment load distribution in tree networks with delays. IEEE Trans. Aerosp. Electron. Syst. **31**(2), 555–567 (1995)
14. Yang, Y., Casanova, H.: Extensions to the multi-installment algorithm: affine costs and output data transfers. Technical Report CS2003-0754, Dept. of Computer Science and Engineering, University of California, San Diego (2003)
15. Millot, D., Parrot, C.: Scheduling on unspecified heterogeneous distributed resources. In: Proceedings of the 25th International Symposium on Parallel and Distributed Processing Workshops (IPDPSW'11), vol. 1, pp. 45–56. IEEE Computing Society Press, May 2011
16. Millot, D., Parrot, C.: Fundamental results on the AS4DR scheduler. Technical Report RR-11005-INF, TELECOM sudParis, Évry, France (2011)
17. Millot, D., Parrot, C.: Some tests of adaptivity for the AS4DR scheduler. In: Proceedings of the 41th International Conference on Parallel Processing (ICPP'12), pp. 323–331. IEEE Computing Society Press, September 2012
18. Pisinger, D.: An exact algorithm for large multiple knapsack problems. Eur. J. Oper. Res. **114**(3), 528–541 (1999)
19. Casanova, H., Legrand, A., Quinson, M.: SimGrid: a generic framework for large-scale distributed experiments. In: Proceedings of the 10th International Conference on Computer Modeling and Simulation (ICCMS'10), pp. 126–131. IEEE Computing Society Press, March 2008

The 5th Workshop on Language-Based Parallel Programming Models (WLPP 2013)

Towards Standardization of Measuring the Usability of Parallel Languages

Ami Marowka(✉)

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
amimar2@yahoo.com

Abstract. The efforts of the research community and the software industry to make the art of parallel programming easier continue. Measuring the usability of contemporary parallel programming languages and libraries by empirical studies is the key to understanding how programmers are thinking, designing, coding, and debugging parallel programs. In this paper we take apart into their component ingredients the empirical experiments done in the recent years. By analyzing each component separately we can better understand what is missing in these experiments and thereby improve the outcome of future studies. The result of this work is a set of recommendations that aims to make usability studies more convincing so that parallel language designers will take them seriously.

Keywords: Usability · Parallel language · Empirical study · Productivity

1 Introduction

The paradigm shift towards many-core computing is irreversible. Parallelism is the only feasible economic solution for increasing performance per watt. While power-efficient many-core processors have become commonplace, development of software for these processors has lagged behind. The main stumbling block for the widespread adoption of parallel computing in mainstream software development is the lack of user-friendly development tools (languages, libraries, debuggers, and performance analyzers) that ease parallel programming especially for inexperienced programmers. Furthermore, adding parallelism can cause new problems to the application such as deadlock, race conditions, synchronization, communication, heterogeneity, load-balancing, and non-determinism. These problems make parallel programming cumbersome and error-prone.

Usability of a parallel language is measured by how easy it is to learn how to design, develop, code, test and debug a parallel program by using the language features [1]. Therefore, it directly affects the productivity of software developers. In the last two decades empirical studies were performed, usually with novice programmers as experiment subjects, for measuring quantitatively the usability of new paradigms and for comparing them. Such empirical studies are crucial tools for understanding what is missing in these paradigms and how we can improve the usage of future languages and tools. However, not enough work has been done to assess the usability of parallel languages or to develop criteria for such assessments. As a result, the usability measurement of a parallel language is usually based on how easily a toy parallel application or, less frequently by the use of a small set of simple parallel algorithms, can be coded as compared to MPI. Without a doubt, we must collect more information on how parallel programming is done and how its practice is affected by different processor architectures, tools, and languages.

We performed an in-depth examination of the field experiments that were conducted during the last two decades for measuring the usability of various parallel languages and libraries. Our main observation from reading these experiment reports was the lack of standardization for measuring the usability of parallel tools. In other words, we found a complete mess in this important research field.

In this paper we present the methodological building blocks of previous empirical experiments. Our intent is to study the pros and cons of each module of these experiments in order to be able to define a standard for parallel usability that will be supported and adopted by the research community. There is no reason why the same community that defined many software standards and standards de facto such as benchmarks for measuring performance of microprocessors and supercomputers cannot agree on a standard for the assessment of language usability.

In particular, we make the following contributions:

1. We identify and describe the modules that comprised the past empirical studies.
2. We recommend how to modify the context of these modules and which modules are missing and must be added.
3. We present an overview of related work on the usability of parallel programming languages.

The remainder of this paper is structured as follows. In Sect. 2 we review the components of past experiments. In Sect. 3 we give an overview of related work. We conclude and present directions for future work in Sect. 4.

Remark: Hereafter we use the term *parallel language* to represent parallel programming models, libraries, or language extensions that enable parallelism in explicit or implicit manners.

2 The Building Blocks of Empirical Experiments

2.1 The Algorithms

A glance over past empirical studies reveals that many different kinds of algorithms were used to evaluate the usability of a specific parallel language. A few of them are simple algorithms, others demand more sophisticated solutions and still others are just toys applications. Some experiments used only one algorithm while others adopted a suite of algorithms wherein each one represented a different design pattern.

Among the single-algorithm experiments are algorithms that compute a transitive closure [4]; that model heat flow between identical blobs connected by rods of varying conductivity [5]; the Smith-Waterman local sequence matching algorithm [6]; the sparse-matrix dense-vector multiplication [14]; a simple string manipulation algorithm [9], and a toy application called sync-gallery [16].

Among the multiple-algorithm experiments one can find the nearest neighbor applications (Game of Life, Grid of Resistors) [13]; Kernels of NAS Parallel Benchmark, GUPS, Histogram and N-Queens applications [15], and the Cowichan Problems [7,8].

There are also others experiments that used only syntactic, pen and paper analysis without empirical experiment [2], or used an analysis of a corpus of many open-source applications developed by unknown many programmers [10].

A single-algorithm suite can assess only a small fraction of the expressive potential of a given language and therefore is not an appropriate solution for usability measurement. A multiple-algorithm suite is the preferred choice. It is advisable that such a suite will contain a set of problems for novice programmers and another set of problems for parallel professional programmers. Moreover, the problems have to represent the most commonly used parallel design patterns. A good starting point for such a suite would be the Cowichan Problems.

The Cowichan Problems are a suite of seven problems (elastic net, Gaussian elimination, halving shuffle, invasion percolation, Game of Life, Mandelbrot Set, and point normalization). These problems cover different aspects of parallelism (data-parallelism vs. task-parallelism) from different angles (regular vs. irregular communication patterns). They were designed to address issues such load-balancing and non-determinism and to represent numerical and symbolic applications. Furthermore, the most interesting point in the design of the Cowichan problems is the fact that they are chainable. This means they can be concatenated to reflect real-life applications where the output of one problem is the input of a second problem.

2.2 The Languages

Many parallel languages, libraries, and language extensions were evaluated in the field experiments. Some of them were designed for shared-memory architectures while others were intended for distributed-memory architectures. But there are languages that were designed for both architectures. Usually the tested languages are research languages that are self-evaluated or compared to a standard de facto language. However, the strangest experiments are those that compared the usability between shared-memory (apples) and message-passing languages (oranges). The results of such a comparison are so obvious and thus biased toward the shared-memory languages. The reason is that the new shared-memory languages offer a higher level of abstraction and thus achieve coding with fewer lines of code (LOC) as compared to an implementation with MPI. Moreover, the downside of higher abstraction, loss of performance, is never measured or mentioned in those experiments.

Among the self-evaluated languages that were studied are Orca [7], Correct Object-Oriented Pattern-based Parallel Programming System (CO2P3S) [8] that implemented the Cowichan problems suite and Microsoft TPL [10]. But most of the experiments are based on comparisons: Enterprise vs. PVM [4], Java implementations of Actor model vs. transactional memory (TM) vs. shared-memory [5], multi-threaded Java vs. SCOOP [9], Java 7 vs. Habanero-Java (HJ) [2], MPI vs. UPC vs. X10 [6], MPI vs. OpenMP [13], MPI vs. XMTC [14], UPC vs. MPI [15], and Java vs. TM [16].

Empirical studies aim to discover unacceptable results and not only to confirm a priori known results. For example, programming with transactional memory (TM) is considered to be easier than programming with locks. Rossbach and Hofmann [16] had very interesting findings. The student-subjects who participated in the experiment claimed that programming with transactions is harder to use than those with coarse-grain locks, but slightly easier to use than those with fine-grained locks. However, an examination of the students' code disclosed that over 70 % of the students made synchronization errors with fine-grained locking as compared to less than the 10 % who made errors with transactions.

2.3 The Hardware Platforms

Surprisingly, most of the empirical studies conducted in the past did not report on the parallel machines that were used during the experiments. It is important to do so for the following reasons:

1. Experiments should be reproducible and without detailed descriptions of the hardware platform or the results will not be comparable. For example, there is a difference between developing and running an algorithm on a dual-core machine and doing so on a 64-core machine especially from scalability point of view. We will elaborate more on this issue later.
2. A language that was designed for Petaflop supercomputers is not like a language that was designed for multicore machines. In order to increase the reader confidence in the report this information is essential.
3. There are experiments that use simulations rather than real machines. In these cases the results should be taken with a grain of salt.

A few experiments mention the hardware platforms. In [7] the authors used a network of 80 workstations, in [4] a network of 50 SUN 4 workstations, and in [8] a SGI Origin 2000 machine with 46 MIPS R100 processors. In [14] the authors compared MPI vs. XMTC. The MPI implementations were run on a 24-processor SUN SunFire system while the XMTC implementations run on prototype compiler simulator software.

2.4 The Parallel Debuggers

Debugging, profiling, and analyzing a parallel program are highly tedious and difficult tasks. Although parallel debuggers, visual profilers, and performance analyzers exist and are improving all the time, finding a bug in a parallel program is like finding a needle in a haystack. The complexity of parallel debugging is due to the invisible problems and to the timing complexity of parallel program flow that harden on finding temporary bugs, whose appearance cannot be predicted. Therefore, using such a tool set is crucial for bug-free programming and an integral part of any developer's tool box.

Unfortunately, none of the experiments reported on the use of such a tool set by the human-subjects.

A standard for measuring the usability of parallel languages must include an examination of how debugging tools were used during the experiment.

2.5 The Human-Subjects

Most of the usability studies were performed in an academic setting as part of a parallel programming course. Usually the human-subjects were graduate students inexperienced in parallel programming. For example, in [7] six graduate student novices in parallel programming participated in the experience; in [9] 67 B.Sc. students participated after taking a course on parallel programming; in [8] the authors performed the experiment by themselves; in [2] the authors performed a pen and paper assessment without human-subjects and in [16] 237 undergraduate students who were inexperienced in parallel programming participated in the experiment.

Recruiting qualified human-subjects for participation in an empirical experiment is a very critical mission because it will determine the final quality of the results. The reports of past field experiments show that there are cases where the participants simply left the lab in the middle of the experiment or failed to complete their tasks in time. In [19] the authors admit that they failed to recruit enough qualified subjects.

2.6 The Metrics

Effective metrics are critical for measuring different aspects of parallel usability. Unfortunately, the metrics that are used today do not yield enough observational insights required for analyzing the data produced during controlled experiments.

The most common, though controversial, metric is the lines of code (LOC). There is a consensus among researchers that the LOC metric is a weak one and that better metrics must be developed. Therefore in [15] the authors added the number of characters (NOC) metric while the conceptual programming effort was measured by the number of parameters passed, the number of keywords used, and the number of function calls and their types. In [7] the authors mention three metrics but do not use them in their analysis: Halstead's "program volume" measure, McCabe's cyclomatic complexity and measures based on decomposition of flow-graphs

There are cases where empirical studies assessed the usability of a given parallel language without using metrics at all, and others that used heuristic metrics.

In [6] the "time to first correct parallel solution" metric was used to measure productivity. This time was further broken down using heuristic algorithms to identify development phases such as authoring, debugging, and program execution. In [16] the metrics used are the time spent designing, coding, debugging, checking the accuracy of the program and how many bugs were found. In [5] three metrics were used: the time taken for subjects to complete the task, the number of (non-comment) lines of code in the final program, and the subjective rating. In [13] the programming effort was measured by the time taken for serial coding, parallelization, debugging, and tuning. The code expansion factor was measured by LOC and cost per LOC (in person-hours).

2.7 The Experiment Duration

Many-core programming demands new programming skills and a different kind of problem-solving. Sequential programming is a deterministic and predictable process. Therefore, it arises intuitively from the way programmers solve problems using algorithms. In contrast, many-core programming is intrinsically a nondeterministic process. Parallelism requires the programmer to think in a way that humans find difficult. However, it is possible to learn how to think in parallel and how to write bug-free parallel programs. In other words, it takes time to teach a new programming paradigm plus a new parallel language plus new debugging tools. Furthermore, it takes time to internalize the new concepts and to be able to parallelize algorithms efficiently for both inexperience programmers and professional programmers. Unfortunately, from the reports of past experiments we are not convinced that the time duration that was given for training and for solving the problems was enough.

From the experiments that reported and recorded times we noticed the following findings. In [7] the experiments took up to 9 months. In [4] a 50 min tutorial was given for each tested language and the experiment's duration was two weeks. In [5] only four hours of training were given to each student. In [9] the training phase ran during a two-hour lecture session which was followed by a test phase of two hours. In citebib13 there were participants that finished the tasks after a few hours and others after a few days. In [6] two days of tutorials were given for each language followed by two days of intense parallel programming. In [13,16] the experiments were conducted as part of a semester course in high performance programming, and in [14] the programmers were given a deadline of two weeks and they were allowed to work on the assignment in their own time.

2.8 Usability vs. Scalability

Usability of a parallel language cannot ignore the aspect of performance. If a given parallel language is easy to learn and enables one to design, code, test and debug a parallel program easily but produces programs that are not scalable then it is useless. Unfortunately, in most of the past experiments the participants were not asked to try to achieve an a priori defined speedup.

New programming frameworks provide high abstractions that are less general but more easily understood. However, abstractions also increase productivity and code correctness at the expense of performance. This trade-off must be measured and weighed accordingly, especially when a comparison is made between a low-level programming model such as MPI and a high-level programming model such as OpenMP.

Since usability comprises the time it takes to achieve a solution with a certain performance, the experiment must define minimum and maximum performance objectives.

2.9 Human vs. Machine

Can off-line and blind analysis done by a machine replace on-line observations of human-subjects in studying the usability of a parallel language?

We do not think so, but there are others who believe it can. For example, Okur and Dig [10] studied the usage of Microsoft's Task Parallel Library (TPL) and Parallel Language Integrated Query (PLINQ). The experiment analyzed a corpus of 655 open-source applications (from Github [17] and Microsoft's CodePlex [18]) developed with C# by 1609 unknown programmers. Such a syntactic and semantic analysis cannot tell anything about the skills of the programmers or interview them in order to understand why they did what they did. The authors' findings raise questions without the possibility of answering them. For example, they found a piece of code that was parallelized by the programmer, who was not aware that the code actually runs serially. However, since the programmer remains anonymous nobody can ask her/him if she/he was aware of the mistake and why she/he did not check that the code indeed runs in parallel. A simple check could immediately reveal that the code runs serially.

We prefer controlled experiments where the tested environment is instrumented to record all editing, compiling, and execution actions. During the experiment the human-subjects are asked to write down comments and can be helped by available assistants. Furthermore, in [5] the subjects' screens were recorded, and a webcam recorded the subject to monitor off-computer events. A snapshot of the project directory was taken at 1-minute intervals, and instrumented tools logged each invocation of the compiler and each run of the resulting program. In [6] the programming activities of the study participants were recorded, both through face-to-face observations by the experiment teams as well as through frequent automated sampling of the programs being written, so that it was later possible to analyze the progress of each study participant in great detail, including the thought processes and the difficulties encountered.

3 Related Work

Sadowski et al. [3] summarize the research that has been done so far by the software engineering community in order to study the usability of parallel programming models and languages. The authors outline the research challenges for increasing productivity of parallel programming languages, such as how to incorporate software maintenance, program correctness, and resilience in the face of failure as part of usability evaluation. Moreover, they highlight five research topics that may improve the usability assessment (programming metaphors, visualization techniques, correctness comprehension, social support, and improved cost models). The authors emphasize an important observation that is worth quoting: "parallel programming is difficult, and... writing a correct parallel program is even more difficult. In fact, writing correct parallel programs may be even more difficult than novice programmers realize". The authors conclude that better metrics and a comprehensive taxonomy of parallel problems are needed for improving the evaluation of parallel programming productivity.

Ebcioglu et al. [6] studied the productivity of three parallel programming models (MPI, UPC, and x10) with 27 novice parallel programmers who were asked to parallelize the Smith-Waterman local sequence matching algorithm. From the analysis of automated and non-automated observations such as sampling of the source code changes, recording of the results of compilation and execution, face-to-face observations, and interviews, the authors observed that the programmers showed an inability to use the right concise abstractions; a lack of programming style; a lack of knowledge of parallel design idioms, and a lack of knowledge dealing with non-deterministic programming.

In [9,11,12], Nanz et al. present the design of a study template for comparing the usability of parallel programming languages. The authors claim that their methodology improves the objectivity of an empirical experiment because it reduces the influence of the trainers on the trainees during the study of a new language (regarding the ways problems should be solved in parallel) and during the examination of the solutions (regarding the subjectivity of the interpretation of the solutions). The template is based on a self-study of the languages to be compared, a self-comparison survey, and a self-evaluation scheme for interpreting the comparison survey. The authors used their approach for comparing the usability of multithreading Java and SCOOP (Simple Concurrent Object-Oriented Programming). Sixty-seven students participated in the experiment and found that SCOOP's usability is better.

In [13] Hochstein et al. studied the programmer productivity with an emphasis on novice HPC programmers. They conducted a controlled empirical study across multiple universities and classes of HPC courses. The results of the field experiments were analyzed in order to understand the impact of various parallel programming models and applications on the effort of the programmer to create an efficient parallel program. The experiments included 69 novice parallel programmers in three classes who were asked to parallelize two applications: the Game of Life and Grid of Resistors using MPI and OpenMP on Network-of-Workstations and shared-memory machine respectively. The analysis of the collected data used three metrics: Effort (time of serial coding, parallelization, debugging, and tuning), code expansion Factor (measured by LOC) and cost per LOC (in person-hours). The authors examined a few well-known hypotheses regarding the developing efforts required to program with MPI and OpenMP and did not find any new exceptional insights.

In [16] Rossbach et al. present an empirical study to measure the usability of traditional programming with locks as compared to programming with transactions. The human-subjects were 237 undergraduate students inexperienced in parallel programming who took 5 classes over 3 semesters during 3 different school years. They were asked to parallelize the same problems (that produced a set of 1323 parallel programs) with the two programming paradigms (in Java). The designers of the experiment evaluated the usability by analyzing the final reports of the students regarding the amount of time they spent designing, coding, and debugging each programming task and by checking the correctness of the programs. The students claimed that programming with transactions is harder to use than coarse-grain locks, but slightly easier to use than fine-grained locks. However, examination of the students' code showed that over 70.

4 Conclusions

A standard benchmark suite for measuring usability is needed to enable a comparison of different parallel programming paradigms. The Cowichan suite can be a good starting point. Implementing such a benchmark will produce a detailed report in order to evaluate the weaknesses and strengths of the tested language. To achieve this goal, new metrics and methods for assessment of the usability of parallel languages have to be developed. Furthermore, empirical studies have to focus first on exploring state-of-the-art parallel programming languages and models before designing new parallel languages.

References

1. Nielsen, J. http://www.useit.com/alertbox/20030825.html
2. Cave, V., Budimlic, Z., Sarkar, V.: Comparing the usability of library vs. language approaches to task parallelism. In: Proceedings of the Evaluation and Usability of Programming Languages and Tools, PLATEAU '10, 18 October 2010, Reno, NV, USA (2010)
3. Sadowski, C., Shewmaker, A.: The last mile: parallel programming and usability. In: FoSER 2010, 7–8 November 2010, Santa Fe, New Mexico, USA (2010)
4. Szafron, D., Schaeffer, J., Edmonton, A.: An experiment to measure the usability of parallel programming systems. Concurr. Pract. Exp. **8**(2), 147–166 (1996)
5. Luff, M.: Empirically investigating parallel programming paradigms: a null result. In: Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) (2009)
6. Ebcioglu, K., Sarkar, V., El-Ghazawi, T., Urbanic, J., Center, P.S.: An experiment in measuring the productivity of three parallel programming languages. In: Workshop on Productivity and Performance in High-End Computing (PPHEC) (2006)
7. Wilson, G.V., Bal, H.E.: An empirical assessment of the usability of Orca using the Cowichan problems. IEEE Parallel Distrib. Technol. **4**(3), 36–44 (1996)
8. Anvik, J., Schaeffer, J., Tan, K.: Why not use a pattern-based parallel programming system? In: Kosch, H., Böszörményi, L., Hellwagner, H. (eds.) Euro-Par 2003. LNCS, vol. 2790, pp. 81–86. Springer, Heidelberg (2003)

9. Nanz, S., Torshizi, F., Pedroni, M., Meyer, B.: Design of an empirical study for comparing the usability of concurrent programming languages. In: International Symposium on Empirical Software Engineering and Measurement. pp. 325–334 (2011)
10. Okur, S., Dig, D.: How do developers use parallel libraries? In: FSE (2012)
11. Nanz, S., Torshizi, F., Pedroni, M., Meyer, B.: Empirical assessment of languages for teaching concurrency: methodology and application. In: Proceedings of the CSEE&T'11, pp. 477–481. IEEE Computer Society (2011)
12. Nanz, S., Torshizi, F., Pedroni, M., Meyer, B.: A comparative study of the usability of two object-oriented concurrent programming languages. http://arxiv.org/abs/1011.6047 (2010)
13. Hochstein, L., Carver, J., Shull, F., Asgari, S., Basili, V.: Parallel programmer productivity: a case study of novice parallel programmers. In: Proceedings of the SC'05, p. 35. IEEE (2005)
14. Hochstein, L., Basili, V.R., Vishkin, U., Gilbert, J.: A pilot study to compare programming effort for two parallel programming models. J. Syst. Softw. **81**(11), 1920–1930 (2008)
15. Cantonnet, F., Yao, Y., Zahran, M., El-Ghazawi, T.: Productivity analysis of the UPC language. In: Proceedings of the IPDPS'04 (2004)
16. Rossbach, C.J., Hofmann, O.S., Witchel, E.: Is transactional programming actually easier? In: Proceedings of the PPoPP'10, pp. 47–56. ACM (2010)
17. GitHub. https://github.com
18. CodePlex Open Source Project Hosting. http://codeplex.com
19. Halverson, C.A., Carver, J.: Climbing the plateau: getting from study design to data that means something. In: Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) (2009)

Experiences with Implementing Task Pools in Chapel and X10

Claudia Fohry[1(✉)] and Jens Breitbart[2]

[1] Research Group Programming Languages/Methodologies, University of Kassel, Kassel, Germany
fohry@uni-kassel.de

[2] Engineering Mathematics and Computing Lab (EMCL), Heidelberg University, Heidelberg, Germany
jens.breitbart@iwr.uni-heidelberg.de

Abstract. The Partitioned Global Address Space (PGAS) model is a promising approach to combine programmability and performance in an architecture-independent way. Well-known representatives of PGAS languages include Chapel and X10. Both languages incorporate object orientation, but fundamentally differ in their way of accessing remote memory as well as in synchronization constructs and other issues of language design.

This paper reports on and compares experiences in using the languages. We concentrate on the interplay between object orientation and parallelism/distribution, and other issues of coding task parallelism. In particular, we discuss the realization of patterns such as objects that internally contain distributed arrays, and suggest improvements such as support for activity-local and place-local data, as well as scalar variable-based reduction. Our study is based on Unbalanced Tree Search (UTS), a well-known benchmark that uses task pools.

Keywords: Chapel · X10 · PGAS · UTS · Task pool

1 Introduction

The goal of high-productivity parallel programming has led to the Partitioned Global Address Space (PGAS) programming model and its concretization in a number of languages/systems such as Chapel and X10. PGAS languages expose to the programmer a shared memory that is split into disjoint partitions. Each partition comprises distinct computing resources that have faster access to local than to remote memory.

Chapel and X10 have similar goals and a similar level of maturity, but differ in many other aspects. Chapel was introduced by Cray, and X10 by IBM, both at the beginning of this century and with funding from the DARPA "High Productivity Computing Systems" project. The paper refers to Chapel version 1.8.0, and X10 version 2.4. A brief survey of the languages is given in Sect. 2.

We base our comparison on Unbalanced Tree Search (UTS), a benchmark for studying issues such as load balancing or characteristics of the implementation language [6]. Unlike previous Chapel and X10 implementations of UTS [3,13], our programs use both multiple nodes and multiple threads per node. UTS is implemented with task pools, deploying a fixed number of long-running workers and a distributed data structure for mutual access. The benchmark and this setting were selected to study communication, synchronization, and the interplay between object-orientation and parallelism/distribution in a simple framework. They may set Chapel at some disadvantage by not appreciating its rich set of built-in data parallel features.

Our focus is expressiveness of the languages, moreover some preliminary performance numbers are included. We report on our experiences in deploying the available language constructs, discuss variants for coding common patterns such as objects that internally contain distributed arrays, and suggest improvements such as support for activity-local and place-local data as well as scalar variable-based reduction. Moreover, we touch on diverse issues such as objects vs. records and locality optimization.

The paper starts with background on Chapel, X10, task pools and UTS in Sect. 2. That section also details the task pool variant that we selected. Section 3 gives an overview of our implementations, and then organizes language assessment along various topics. Thereafter, Sects. 4, 5 and 6 are devoted to performance, related work and conclusions, respectively.

2 Background and Benchmark

2.1 Chapel

The following introduction is by necessity brief, for further information see [2].

A Chapel program runs on some number of *locales*, each of which comprises processors and a memory partition. The `on` statement places a code block on a particular locale. Within the block, all variables in scope may be read and written, although accesses to remote locales are more expensive.

Parallel tasks are created with, e.g., `begin` or `coforall`. Task creation can be combined with code placement, for an example see Sect. 3.4. Chapel does not expose threads, but tasks are transparently mapped to threads by a configurable tasking layer.

In Chapel, synchronization is almost exclusively based on synchronization variables, which are declared with type qualifier `sync` or `single`. The former hold a value of some primitive type, and additionally have state full or empty. A write to a full `sync` variable blocks the calling task, as does a read from an empty one. When the variable changes state, one of the waiting tasks may proceed.

The base language has C-like syntax. Arrays are defined over multidimensional domains and may be distributed, e.g. blockwise or cyclic. Constants may be marked as configurable, in this case they may be overwritten at the command line.

Chapel programs are composed of modules, which contain data, functions, classes etc. Classes have the usual functionality, including constructors, inheritance, nesting, and generics. Records resemble classes, but variables of this type directly hold the values of all fields.

2.2 X10

Many Chapel concepts have an analogue in X10 [17], except for using different terminology:

Chapel	locale	domain	`on`	task	`begin`	`record`	`sync` statement
X10	place	region	`at`	`activity`	`async`	`struct`	`finish`

The concrete realization differs, e.g. X10 structs are less flexible than Chapel records. Unlike in Chapel, variables may only be accessed if they are stored at the current place, are globally accessible through a `GlobalRef`, or have been copied with `at`.

`at` copying rules are complex, yet in general single-assignment variables (specified with `val`) are copied, whereas normal variables (specified with `var`) are not. The X10 standard library supports place-local data, which are accessed through a `PlaceLocalHandle` that may be communicated and resolved at any particular place.

The major synchronization construct, `atomic`, encloses a critical section and operates intra-place. All read and write accesses to shared variables must be protected, and all critical sections at a place are mutually exclusive.

In many respects, the base language resembles Java. We used the most elementary type of arrays, called rails. There is no equivalent of a Chapel module, but classes may have static fields, and support inheritance, nesting, and generics.

2.3 Task Pools

Many irregular applications are composed of sub-computations (tasks) that vary in size. Task pools are a well-known pattern to map these tasks to execution resources at runtime, and thereby achieve load balancing. Task Pools may be implemented in either the user program or the runtime system, and come in various forms. In the following, we only describe the variant that we implemented, which resembles the one in [10]. Note that the task pool literature uses the term task different from PGAS languages. To avoid confusion, we denote the execution resources, which will correspond to X10 activities or Chapel tasks, as workers.

A task pool is a data structure from which idle workers repeatedly take a task, compute it, possibly insert new tasks, take the next task etc., until the pool is empty. The data structure is distributed. Each worker maintains a split queue [10], which is a kind of circular buffer that comprises a private and a public portion. It is double-ended, such that `head` denotes the first free position of the private part, and `tail` the last filled position of the public part. The elements in-between `head` and `tail` are divided into `nprivate` elements in the private pool, followed by `npublic` elements in the public pool.

The task pool is accessed by `push` and `pop` operations: `push` inserts a task at position `head`, and `pop` takes a task out from the same end. If the private pool holds $2k$ elements, for some constant k, `push` additionally `releases` k elements to the public pool. Analogously, if `pop` discovers an empty private pool, it `acquires` k elements from the public pool. Operations `acquire` and `release` do not move tasks, but shift the division line between the private and public portions. Synchronization is required for the public pool only.

When `acquire` fails, the worker tries to steal k tasks from some other worker. Therefore, it first cycles through all workers of its own place, and then through those of the others. When no victim is found after one global cycle, the worker terminates. Termination detection is actually more complex [11], but we rely on the simple scheme for brevity.

2.4 UTS

A task pool may either be provided as a reusable component, e.g. by a library, or be used as a pattern to implement a particular algorithm. We considered the second scenario with the Unbalanced Tree Search (UTS) benchmark [6].

UTS consists in extracting a tree and counting the number of nodes. For given tree shape parameters, a node holds all information about the subtree rooted in it, and thus may be deleted after having been expanded. The information is encoded in a 20-byte node descriptor, using some cryptographic method. Naturally, a task corresponds to the expansion of one node, and is represented by this node descriptor.

Open-source implementations of UTS are available for various systems, among them Chapel and X10 [3,6,13–15]. They will be discussed in Sect. 5. We reused parts of the implementations [3,15], chiefly the respective native interfaces to the C cryptographic tools, and the deployment of place local handles from [15]. Our own code can be obtained from the first author's homepage.

3 Language Assessment

3.1 Overview of Implementations

In both languages, we implemented UTS with the task pool variant described in Sect. 2.3. Our implementation uses both multiple places and multiple activities per place. Workers are realized by long-running activities/Chapel tasks that are started after the task pool has been filled with initial tasks. The functional components of all program variants are similar:

– setting parameters of the tree
– generating and initializing the distributed data structure for the task pool

- expanding the root and inserting initial tasks into the task pool
- managing the workers that process the tasks
- managing the split queues with `push`, `pop`, `release` and `acquire`
- realizing the `steal` operation, including definition of the cyclic order and moving tasks to another place
- computing the result number of nodes by reduction
- invoking C functions for initializing the root and decoding node descriptors.

The X10 standard library includes basic local data structures, but neither Chapel nor X10 provide split queues, which therefore had to be implemented manually. For simplicity, we assume that the public pools never overflow.

3.2 Object-Orientation and Parallelism

The Encapsulation Problem. In object-oriented programming, data structures are often coded as classes, such that each instance of the class represents an instance of the data structure. The reference to this instance is stored in a variable, and operations are invoked by method calls on this variable. Thus, the variable provides a single access point to the data structure, abstracting away all details of the internal representation at the caller site:

```
var s: Stack = new Stack();
s.push(elem);
```

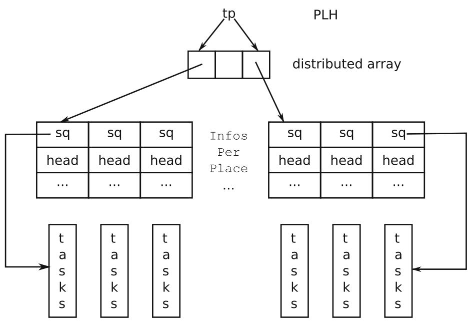
When the data structure is distributed and accessed from different places, such as our task pool, there is currently no equivalent for this convenient notation, since a single access point `s`, located in a single place, hurts performance.

To solve the problem, we distributed the data structure across places and addressed the local portions. We implemented several variants of this "distributed-first" approach, which all share the drawback that they provide less encapsulation than the above "objects-first" approach. In particular, they do not hide, at the caller site, the fact that the task pool is distributed. Thus, a programmer must decide between either the well-structured but inefficient "objects-first", or the more efficient but less modular "distributed-first" approach.

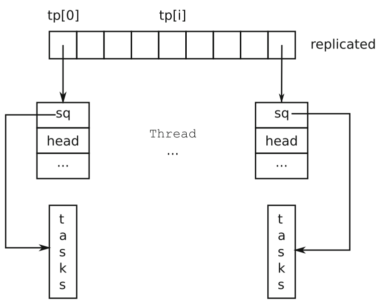
X10 Implementation. We start the discussion of"distributed-first" variants with X10, since X10 provides some library support with its `PlaceLocalHandle` (PLH). As explained in Sect. 2.2, a PLH supports access to place-local data, and thus simplifies addressing the local portion when invoking task pool operations. Nevertheless, declaration and initialization of the distributed structure remain the responsibility of the user:

```
val tp = PlaceLocalHandle.make[InfosPerPlace](...)
```

In this code fragment, `InfosPerPlace` is a user-defined class which, as illustrated in Fig. 1(a), chiefly contains a rail of split queues. As can be seen in the figure,



(a) Pool structure in X10



(b) Pool structure in Chapel

Fig. 1. Implementation variants

use of the PLH requires a two-level addressing scheme, in which the PLH resolves to place-local information, and then a particular split queue is selected.

A split queue object encapsulates all data of a worker and the respective methods, which has the advantage that split queue methods can directly access the local data through instance fields. Access to remote queues goes through variable `tp`, which is stored in the split queue object.

Chapel Variant. As Chapel does not support a PLH-like construct, we had to explicitly work with a distributed (or replicated) array. We use a one-level addressing scheme, i.e., we have one entry per worker (as opposed to per locale). The Chapel variant is illustrated in Fig. 1(b). Access to remote queues is enabled by declaring the distributed array at module scope.

Place-Local and Activity-Local Data. Comparing the two variants, the PLH concept is appealing since it eliminates the need for explicit indexing. When place-internal indexing is needed instead, the advantage is, however, lost. For our application, PLH-like support for activity-local data would have been most useful. It would have enabled a similar program structure as in Fig. 1(b), but with simpler addressing. While our use of activity-local data is to some degree application-specific, long-running activities reduce the overhead, e.g. for initializing data structures, and may therefore be valuable beyond our application. As an application may mix place-local and activity-local data, we suggest to support both in both languages.

3.3 References, Values, and Copying

In both languages, a tree node may be represented by either an object, or a record (struct). From a performance point of view, records may be cheaper as they do not incur the typical indirection and method resolution costs of objects. Moreover, memory allocation is cheaper if one large block is allocated for all records in the task pool, as compared to allocating pieces of memory for each node object. At the backside, record assignment and parameter passing by value, e.g. in `push`, involve expensive copying and should therefore be avoided.

In addition to class-based versions, we implemented record-based versions that avoid copying by expanding children directly into their task pool entry. A problem arises during the generation of initial tasks, when a child is to be expanded into a remote queue. Chapel calls a C function to decode a node descriptor, which takes as arguments pointers to both the parent and future child descriptors, but it is not possible to pass a pointer to remote memory to this function. Therefore, the parent descriptor is first copied to a remote variable, and then the C function is called. This shows up limits of native code integration with Chapel.

In the X10 program, remote access to native code was easier. Since the reused code from [15] represents a node by a C++ object, that object is automatically copied to the remote place when its native code is required. The approach appears easier but less efficient, especially as X10 generates a deep copy and inclusion of fields can not be controlled at the C++ side.

3.4 Worker Management and Initialization

The base functionality for starting worker tasks is obvious, e.g.:

```
coforall loc in Locales do on loc
  coforall tid in 0..#numWorkersPerLoc do runWorker(tid);
```

The corresponding X10 code is slightly longer, as there is no equivalent of `coforall`, and `async` must be ended by `finish`. Activity-local data would help managing the task identifier `tid`.

To allow stealing, the above `coforall` loop may only be entered after the task pool has been initialized, especially references to the remote split queues must have been set. Task pool initialization is by itself distributed, but the pool must not be used before initialization has finished. This can be achieved, e.g., by a barrier. There are various opportunities to implement this barrier, e.g. in Chapel pairwise synchronization before a worker's first steal access to a victim locale can be implemented with synchronization variables.

3.5 Reduction

Reduction is a well-known pattern to combine multiple values. It can be efficiently parallelized, in our setting by first combining the values within each worker, then within each place, and finally within the overall program. UTS uses reduction to compute the result number of nodes.

Chapel and X10 provide language support for reduction only on the assumption that the values are stored in an array, which has several drawbacks:

- The values must be kept in an array even if it does not match the application's structure. In our Chapel variant, e.g., a local value would logically belong to the `Worker` class, and thus the array needs to be defined and filled just for the purpose of reduction.

– In the array, false sharing between neighbored values is likely.

OpenMP defines reduction differently [7]: A user program declares a scalar variable with some keyword for reduction, and specifies the operator. The system transparently defines local copies, collects values locally, and synchronizes the update of the global result. At least in Chapel, a similar scheme should be possible and would be desirable from a user's point of view.

3.6 Diverse Language Issues

Split Queue Synchronization. Synchronization is required for the public pool. The `steal` critical section, e.g., includes checking `npublic`, modifying `npublic/tail`, and copying the tasks out of the pool. It is kept short by first making a local copy of the tasks, and sending it to the remote place after the critical section. X10 critical sections are coded with `atomic`, whereas our Chapel programs use a synchronization variable that holds `npublic`.

Remote Access. A Chapel programmer may inadvertently access remote variables. Tool support might help and, unlike X10's `at`, not impose any restrictions. When using a PLH, a similar problem occurs when the user forgets `at` as in `for <allPlaces> { tp().init(); }`, where `tp` is always evaluated at the origin. An X10 `at` only copies `val`'s. When they need to be computed before being sent, both a `var` and a `val` variable for the same purpose are needed, which blows up the code and requires copying.

Constants and Parameters. In Chapel, tree parameters are naturally stored in configurable variables that can be easily overwritten on the command line. Parameter passing in X10 is more complicated.

In Chapel, `val`-like variables may be declared with `single`, i.e., there is some redundancy between `const` and `single`. Possibly, `const` may be removed if the `config` label is extended to `single`, and `single` values are replicated across places.

Language vs. Library. By releasing central functionality to the library, chances for integration may be dismissed. In X10, for instance, a language construct such as `at(allPlaces)` would appear elegant.

4 Performance

We run experiments on a cluster of 8-core Intel Xeon E5–2670 processors, with 2 processors per node and Infiniband network. For X10, compiler option `-O` was used, and for Chapel `gasnet/ibv` for multi-node and `none` for single-node execution.

Table 1 shows running times for the `T1L` sample of UTS [14], which is a geometric tree with branching factor 4 and maximum depth 13. The results

Table 1. Running times of different program versions (averaged over 3 runs).

	Chapel class	Chapel struct	X10 class	X10 struct
1 Place 1 Thread	72.8	365.5	44.8	36.4
1 Place 4 Threads	26.6	138.8	16.9	12.6
1 Place 16 Threads	23.5	65.7	7.3	4.5
4 Places 4 Threads	143.0	1286.0	6.1	5.4

suggest a performance advantage of X10 over Chapel. In X10, the struct-based variant was slightly faster than the class-based one, while in Chapel the record-based variant were inferior.

The results can only be considered a snapshot, as the current versions of the languages do not yet exhaust their performance potential. For instance, the release notes state that Chapel 1.8.0 is not suitable for in-depth performance comparisons, and the X10 atomic sections induce unneeded serialization. Most of all, we did not tune the performance, and therefore there is likely much room for improvements in all versions.

5 Related Work

As mentioned in Sect. 2.4, UTS has already been implemented with Chapel and X10. The previous Chapel implementation [3] starts with one task queue. As soon as it has reached a certain size, it is split into two queues, and a new Chapel task is started to process the second queue. The program runs within a single place only. The previous X10 implementation [13] focuses on termination detection, and performance tuning includes low-level functionality such as `IndexedMemoryChunk`. This way, it achieves excellent and scalable performance. In contrast, we took the position of a high productivity programmer and did not tune the performance. The previous X10 implementation deploys only one activity per place and a cooperative work stealing algorithm that does not require synchronization. Unlike these implementations, we closely followed the traditional task pool pattern.

Beyond UTS, several experience reports on coding applications with Chapel and X10 have been published. Referring to older language versions, Shet et al. [12] discuss experiences with a quantum chemistry kernel. Their work includes a central task pool, which is simpler than ours. Weiland [16] presents a nice comparative survey of language features in earlier language versions. Khaldi et al. [4] compare six parallel languages and discuss aspects of expressiveness such as synchronization constructs with the Mandelbrot example. Several recent papers report on experiences in coding applications such as constraint-based local search and the fast multipole method in X10 [9].

While we have used task pools as a benchmark for language design, Chapel and X10 also deploy task pools in the runtime system, to map activities/tasks to threads, see e.g. [5]. Problems of combining object orientation and parallelism have been discussed since a long time [1,8]. This paper focused on encapsulation, and was specific to the PGAS setting.

6 Conclusions

This paper has evaluated Chapel and X10 from a user's perspective, working out both differences and common grounds such as difficulties in integrating object orientation and parallelism. We suggested several modifications to strengthen the languages such as support for place-local and activity-local data, scalar variable-based reduction, and the omission of `const`.

Our work was based on a single benchmark, with focus on task parallelism and object orientation. Before drawing conclusions on the usability of the languages in general, one needs to consider more benchmarks and put a stronger emphasis on performance.

References

1. Agha, G., Wegner, P., Yonezawa, A. (eds.): Research Directions in Concurrent Object-Oriented Programming. MIT Press, Cambridge (1993)
2. Chapel Language Specification, Version 0.94. http://chapel.cray.com/papers.html (2013)
3. Dinan, J., et al.: Unbalanced Tree Search (UTS) benchmark in Chapel. Program source https://chapel.svn.sourceforge.net/svnroot/chapel/trunk/test/studies/uts/ (2007)
4. Khaldi, D., Jouvelot, P., Ancourt, C., Irigoin, F.: Task parallelism and data distribution: an overview of explicit parallel programming languages. In: Kasahara, H., Kimura, K. (eds.) LCPC 2012. LNCS, vol. 7760, pp. 174–189. Springer, Heidelberg (2013)
5. Kumar, V., et al.: Work-stealing by stealing states from live stack frames of a running application. In: Proceedings of the ACM SIGPLAN X10 Workshop (2011)
6. Olivier, S., Huan, J., Liu, J., Prins, J., Dinan, J., Sadayappan, P., Tseng, C.-W.: UTS: an unbalanced tree search benchmark. In: Almási, G., Caşcaval, C., Wu, P. (eds.) LCPC 2006. LNCS, vol. 4382, pp. 235–250. Springer, Heidelberg (2007)
7. OpenMP Application Program Interface, Version 3.1. http://www.openmp.org (2011)
8. Philippsen, M.: A survey on concurrent object-oriented languages. Concurr. Pract. Exp. **12**(10), 917–980 (2000)
9. Publications Using X10. http://x10-lang.org (2013)
10. Ravichandran, K., Lee, S., Pande, S.: Work stealing for multi-core HPC clusters. In: Jeannot, E., Namyst, R., Roman, J. (eds.) Euro-Par 2011, Part I. LNCS, vol. 6852, pp. 205–217. Springer, Heidelberg (2011)
11. Saraswat, V., et al.: Lifeline-based global load balancing. In: Proceedings of the ACM Symposium on Principles and Practice of Parallel Programming, pp. 201–212 (2011)
12. Shet, A.G., et al.: Programmability of the HPCS languages: a case study with a quantum chemistry kernel. In: Proceedings of the International Parallel and Distributed Processing Symposium. IEEE (2007)

13. Tardieu, O., et al.: X10 for productivity and performance at scale: a submission to the 2012 HPC Class II challenge. In: Proceedings of the SC Conference on High Performance Computing, Networking, Storage and Analysis. http://x10-lang.org (2012)
14. UTS. http://hpcrl.cse.ohio-state.edu/wiki/index.php/UTS
15. X10 Code for UTS. http://x10.svn.sourceforge.net/viewvc/x10/benchmarks/trunk/UTS/
16. Weiland, M.: Chapel, Fortress and X10: novel languages for HPC. Technical report, HPCx Consortium (2007)
17. X10 Language Specification, Version 2.4. http://x10-lang.org (2013)

Parampl: A Simple Approach for Parallel Execution of AMPL Programs

Artur Olszak[1(✉)] and Andrzej Karbowski[2,3]

[1] Institute of Computer Science, Warsaw University of Technology, Warsaw, Poland
A.Olszak@ii.pw.edu.pl
[2] Institute of Control and Computation Engineering, Warsaw University of Technology, Warsaw, Poland
A.Karbowski@elka.pw.edu.pl
[3] NASK, Research and Academic Computer Network, Warsaw, Poland

Abstract. Due to the physical processor frequency scaling constraint, current computer systems are equipped with more and more processing units. Therefore, parallel computing has become an important paradigm in the recent years. AMPL is a comprehensive algebraic modeling language for formulating optimization problems. However, AMPL itself does not support defining tasks to be executed in parallel. Although in last years the parallelism is often provided by solvers, which take advantage of multiple processing units, in many cases it is more efficient to formulate the problem in a decomposed way and apply various problem specific enhancements. Moreover, when the number of cores is permanently growing, it is possible to use both types of parallelism.

This paper presents the design of *Parampl* - a simple tool for parallel execution of AMPL programs. *Parampl* introduces explicit asynchronous execution of AMPL subproblems from within the program code. Such an extension implies a new view on AMPL programs, where a programmer is able to define complex, parallelized optimization tasks and formulate algorithms solving optimization subproblems in parallel.

Keywords: AMPL · Parallel · Optimization · Modeling languages

1 Introduction

In recent years, due to the physical processor frequency scaling constraint, the processing power of current computer systems is mainly increased by employing more and more processing units. As hardware supported parallelism has become a standard nowadays, parallel computing and parallel algorithms are recently much of interest. In this paper, we focus on solving optimization problems and defining such problems using AMPL [1]. AMPL - A Modeling Language for Mathematical Programming is a comprehensive algebraic modeling language for linear and nonlinear optimization problems with continuous and discrete variables. AMPL allows to express an optimization problem in a declarative way, very similar to its mathematical form. However, despite being a declarative language, AMPL also allows the constructions present in procedural languages which allow to define the program flow - assignments, conditional expressions and loops. Thus, AMPL makes it possible to define a program that solves multiple problems sequentially and calculates the result based on the solutions of the subproblems. However, processing of the subproblems cannot be explicitly parallelized. For individual problems, the parallelism is often provided by solvers, which take advantage of multiple hardware processing units and employ multithreading when solving optimization tasks. Parallel solvers can greatly improve the performance of solution calculation, utilizing opportunities for parallelism to make use of all available hardware processing units. However, in many situations, it is more efficient to formulate the problem itself in a decomposed way taking advantage of the problem structure and apply various problem specific enhancements and heuristics, which may not be obvious for the solvers or impossible to recognize at all, e.g. applying Benders decomposition or Lagrangean relaxation [2].

In this paper, we present *Parampl*, a simple tool for parallel execution of AMPL programs. *Parampl* introduces a mechanism for explicit parallel execution of subproblems from within the AMPL program code. The mechanism allows dispatching subproblems to separate threads of execution, synchronization of the threads and coordination of the results in the AMPL program flow, allowing a modeler to define complex parallel algorithms solving optimization problems as subtasks.

The rest of this paper is organized as follows. Section 2 describes the related work. Section 3 presents the design of *Parampl*, including a brief introduction to the usage of *Parampl* in AMPL programs. The evaluation of *Parampl* and experimental results are presented in Sect. 4. Section 5 concludes the study.

2 Related Work

There have been a few works related to extending algebraic modeling languages with constructs allowing defining optimization problems in a decomposed way. One of the solutions is Kestrel [3]. Kestrel is an application that imitates a solver and submits an optimization job to be executed on a remote machine (NEOS server). In AMPL program, Kestrel is chosen as a solver (instead of the solver name). The remote solver and other Kestrel parameters are specified within the *kestrel_options* option while the NEOS server IP address and port are specified by the *neos_server* option:

```
# kestrel instead of the solver name
option solver kestrel;

# configuration of kestrel:
option kestrel_options 'solver=<solverName>';
option neos_server 'www.neos-server.org:3332';
```

The optimization task is then submitted for remote execution by a regular call of *solve* command. Upon receiving a task, the NEOS server returns the job number and the job password:

```
Job has been submitted to Kestrel Kestrel/NEOS Job number: 6893
Kestrel/NEOS Job password: FaahsrIh
```

Depending on the mode of operation, Kestrel either waits for the NEOS Server to send back the solver's results (blocking) or the *solve* command call returns immediately, in which case the returned job number and password allow to retrieve the solution for a previously submitted job:

```
# configure Kestrel to continue the previously submitted job:
option kestrel_options 'job=6893 password=FaahsrIh';
solve;
```

However, such a solution is sometimes not flexible enough - it depends on the NEOS server and to submit multiple parallel tasks to it (without any extension), it requires user interaction. Thus, in our opinion, *Parampl* is a much simpler and more convenient alternative to the Kestrel interface, as *Parampl* allows to submit multiple problems for parallel execution and retrieve the solutions within the program code, without the need of any user interaction.

A very interesting approach was presented in [4]. The authors present a structure-conveying algebraic modeling language for mathematical and stochastic programming (SML). The language is an extension of AMPL which allows definition of the model from sub-models. The main extension over AMPL is the introduction of the *block* keyword used to define sub-models. The *block* sections group together sub-model entities and allow them to be repeated over an indexing set:

```
block nameofblock {j in nameofset} : {
    ...
}
```

The blocks may contain any number of *set*, *param*, *subject to*, *var*, *minimize* or nested *block* definitions. Such an extension allows the modeler to express the nested structure of the problem in a natural and elegant way. The solution is generic as the block structure is passed to the solvers within the problem definition file, so SML can be used with any structure-exploiting solver.

SET [5] is another approach which allows defining the structure of the problem in a separate structure file. In [6] the authors prove that AMPL's declared suffixes can be used to define the structure of the problem in many common situations. Furthermore, several approaches targeted at stochastic programming have been proposed, for example sMAGIC [7], SAMPL [8], and StAMPL [9].

In this paper, we present a different approach that enables a modeler to define a fork-join structure of the program flow, allowing processing the results of the subtasks by a coordination algorithm. Our solution is not dependent on the solver used and the parallel execution and results retrieval is handled on the AMPL level by the modeler.

3 Design of Parampl

Let us consider a very simple AMPL program, which solves sequentially the same problem for two sets of parameters p1, p2 and stores the results in one vector *res*:

```
var x{i in 1..3} >= 0;

param res {i in 1..6}
param p1 {iter in 1..2};
param p2 {iter in 1..2};
param iter;

minimize obj:
   p1[iter] - x[1]^2 - 2*x[2]^2 - x[3]^2 - x[1]*x[2] - x[1]*x[3];

subject to c1:
   8*x[1] + 14*x[2] + 7*x[3] - p2[iter] = 0;

subject to c2:
   x[1]^2 + x[2]^2 + x[3]^2 -25 >= 0;

let p1[1] := 1000;
let p1[2] := 500;
let p2[1] := 56;
let p2[2] := 98;

for {i in 1..2} {
   # Define the initial point.
   let {k in 1..3} x[k] := 2;

   let iter := i;

   solve;

  #store the solution
   for {j in 1..3} {
      let res[(i-1)*3 + j] := x[j];
   };
};

display res;
```

Individual calls of the *solve* command will block until the solution is calculated. Using *Parampl*, it is possible to solve multiple problems in parallel. *Parampl* is a program written in Python programming language, which is accessed from AMPL programs by calling two AMPL commands:

- *paramplsub* - submits the current problem to be processed in a separate thread of execution and returns:

```
write ("bparampl_problem_" & $parampl_queue_id);
shell 'python parampl.py submit';
```

- *paramplret* - retrieves the solution (blocking operation) of the first submitted task, not yet retrieved:

```
shell 'python parampl.py retrieve';
if shell_exitcode == 0 then {
  solution ("parampl_problem_"& $parampl_queue_id &".sol");
  remove ("parampl_problem_"& $parampl_queue_id &".sol");
}
```

The *paramplsub* script saves the current problem to a *.nl* file (using AMPL command *write*) and executes *Parampl* with the parameter *submit*. When executed with the parameter *submit*, *Parampl* creates a unique identifier for the task, renames the generated *.nl* file to a temporary file and executes a solver in a separate process passing the problem file to it. Information about the tasks being currently processed by *Parampl* is stored in the *jobs* file - new tasks are appended to this file. The tasks submitted in this way are executed in parallel in separate processes. After calculating the solution, the solver creates a *.sol* file with the file name corresponding to the temporary problem file passed to the solver upon execution. The solution may be afterwards passed back to AMPL by calling the *paramplret* script.

The *paramplret* script executes *Parampl* with the parameter *retrieve*, which is a blocking call, waiting for the first submitted task from the *jobs* file (not yet retrieved) to finish - a notification file is generated. The solution file is then renamed to the *.sol* file known by the *paramplret* script and is then passed to AMPL using AMPL command *solution*. At this point, the temporary *.nl* file is deleted and the job id is removed from the *jobs* file. After calling the script *paramplret*, the solution is loaded to the main AMPL program flow as if the *solve* command was called.

The problem presented above may be run in parallel using *Parampl* in the following way:

```
for {i in 1..2} {
    # Define the initial point.
    let x[1] := 2;    let x[2] := 2;    let x[3] := 2;

    let iter := i;

    # execute solver (non blocking execution):
    commands paramplsub;
};
```

```
# the tasks are now being executed in parallel...

for {i in 1..2} {
    # retrieve solution from the solver:
    commands paramplret;

    #store the solution
    for {j in 1..3} {
        let res[(i-1)*3 + j] := x[j];
    };
};
```

In the above scenario, both problems are first submitted to *Parampl*, which creates a separate process for solving each of them (parallel execution of the solvers). In the second loop, the solutions for both subtasks are retrieved back to AMPL and may be then processed.

Before calling the *Parampl* scripts, the parampl must be configured within the AMPL program - the solver to be used and the queue_id should be set. The queue_id is the unique identifier of the task queue, which is a part of the names of temporary files created by *Parampl*, which allows executing *Parampl* in the same working directory for different problems and ensures that the temporary problem, solution and jobs files are not overwritten. The options for the chosen solver should be set in the standard way, e.g.:

```
option  parampl_options  'solver=ipopt';
option  parampl_queue_id  'powelltest';

option  ipopt_options  'mu_init=1e-6 max_iter=10000';
```

4 Evaluation and Experiments

The efficiency of *Parampl* was evaluated on a machine equipped with Intel Core i7–2760QM processor with all 4 cores enabled and Intel SpeedStep, C-States Control, Intel TurboBoost and HyperThreading technologies disabled. The machine was running Windows 7 64-bit operating system, AMPL ver. 20130704 and Python ver. 3.3.2.

The application tested was a decomposed version of generalized problem 20 presented in [10], formulated below:

$$\min_{y \in \mathbb{R}^n} 0.5 \cdot (y_1^2 + y_2^2 + \ldots + y_n^2)$$

$$y_{k+1} - y_k \geq -0.5 + (-1)^k \cdot k, \quad k = 1, \ldots, n-1$$

$$y_1 - y_n \geq n - 0.5$$

The decomposed algorithm divides the vector $y \in \mathbb{R}^n$ into p equal parts (assuming that p is a divisor of even n). Let us denote:

$$x_{i,j} = y_{(i-1)\cdot n_i + j},\ i = 1, \dots, p,\ j = 1, \dots, n_i,$$

$$x_i = \left[x_{i,1}, x_{i,2}, \dots, x_{i,n_i}\right]^T$$

$$\left[x_1^T, x_2^T, \dots, x_p^T\right]^T = y$$

where $n_1 = n_2 = \ldots = n_p = \frac{n}{p}$. We may then divide all n constraints into $p+1$ groups: constraints dependent only on x_i subvector for every $i = 1, \dots, p$:

$$y_{k+1} - y_k \geq -0.5 + (-1)^k \cdot k,\ \ k = (i-1) \cdot n_i + j,\ j = 1, \dots, n_i - 1$$

that is

$$x_{i,j+1} - x_{i,j} \geq -0.5 + (-1)^{k(i,j)} \cdot k(i,j),\ j = 1, \dots, n_i - 1,$$

where $k(i,j) = (i-1) \cdot n_i + j$, and p constraints involving different subvectors x_i and x_{i+1}:

$$y_{k+1} - y_k \geq -0.5 + (-1)^k \cdot k,\ k = i \cdot n_i,\ i = 1, \dots, p-1$$

$$y_1 - y_n >= n - 0.5$$

The latter constraints may be written as:

$$x_{mod(i,p)+1,1} - x_{i,n_i} \geq c_i,\ \ i = 1, \dots, p$$

where

$$c_i = -0.5 + (-1)^{(i \cdot n_i)} \cdot (i \cdot n_i)$$

We define the dual problem as:

$$\max_{\lambda \geq 0} \min_{x_i \in X, i=1,\dots,p} L(x, \lambda) \tag{1}$$

where

$$L(x,\lambda) = \sum_{i=1}^{p} \sum_{j=1}^{n_i} 0.5 \cdot x_{i,j}^2 + \sum_{i=1}^{p} \lambda_i \cdot \left(c_i + x_{i,n_i} - x_{mod(i,p)+1,1}\right)$$

$$= \sum_{i=1}^{p} \left(\left(\sum_{j=1}^{n_i} 0.5 \cdot x_{i,j}^2 + \lambda_i \cdot x_{i,n_i} - \lambda_{mod(p-2+i,p)+1} \cdot x_{i,1} \right) + \lambda_i \cdot c_i \right)$$

The inner optimization in (1) decomposes into p local problems:

$$\min_{x_i \in X} \sum_{j=1}^{n_i} 0.5 \cdot x_{i,j}^2 + \lambda_i \cdot x_{i,n_i} - \lambda_{mod(p-2+i,p)+1} \cdot x_{i,1} \tag{2}$$

Table 1. Simulation results: 4 cores, $n = 6720$

p	Sequential *solve* [s]	Sequential *Parampl* [s]	Parallel *Parampl* [s]	Speedup	Overall speedup
1	1126.9	1143.4	—	—	—
2	873.7	890.9	525.3	1.66	2.15
3	580.7	580.9	225.8	2.57	4.99
4	793.9	801.4	252.3	3.15	4.47
5	707.9	709.0	228.6	3.10	4.93

which may be solved independently, if possible, in parallel. The external - dual problem (the coordination problem) may be solved in the simplest case by the steepest ascent gradient algorithm (iterative):

$$\lambda_i := \lambda_i + \alpha \cdot \left(c_i + \hat{x}_{i,n_i}(\lambda) - \hat{x}_{mod(i,p)+1,1}(\lambda)\right), \quad i = 1, 2, \ldots, p$$

where α is a suitably chosen step coefficient and $\hat{x}(\lambda)$ is the optimal vector built of solutions of local problems (2). The algorithm terminates when no significant change of the result vector is achieved.

For the simulations, we used the solver *IPOPT* [11] ver. 3.11.1 with problem scaling disabled (nlp_scaling_method=none), initial value for the barrier parameter mu_init=1e-6 and the maximum number of iterations max_iter=10e6. Three variants of the algorithm were tested - sequential (solving the subproblems by calling the blocking *solve* command), sequential *Parampl* (using *paramplsub* and *paramplret* calls) and parallel (in every iteration, *paramplsub* was first called for all the subproblems, after which the results were retrieved by calling *paramplret*). The results of simulations for $n = 6720$ and various values of p are presented in Table 1. The column “speedup” presents the speedup achieved when compared to the sequential execution of the decomposed algorithm while “overall speedup” is the speedup in comparison to the calculation time for the original problem.

The decomposed algorithm that we tested appeared to be very sensitive to the problem size (the efficiency varies significantly for various values of n and p - for some problems, more iterations are needed to achieve the same accuracy of the results). It is however a very good example to demonstrate the effect of running programs in parallel on many cores. In the presented simulation results, the effect of employing the parallelism is clearly visible. The larger the number of subtasks, the larger speedup is achieved. The values of speedup are however lower than their upper limit (Amdahl's law), which is caused by the differences of solving times for individual subproblems[1]. Thus, if the differences between computation times for individual subproblems might be significant, the number of parallel subproblems should be greater than the number of physical processor

[1] The time of calculations of the AMPL mathematical instructions in the sequential part, i.e. the time of execution of the coordination algorithm is rather negligible. However, for much smaller problems and large numbers of subproblems and iterations, we noticed that significant portion of the execution time is the startup time of the Python virtual machine and thus the speedup drops sharply.

cores available to minimize the relative time when some cores are idle while waiting for the remaining tasks to complete. It is worth mentioning that the overall speedup reached (compared to the original problem calculation time) is even greater than the number of cores, which was achieved by applying a problem specific heuristic (although the accuracy might be slightly worse). Such a speedup could not be achieved just by employing a parallel solver nor any universal automated tool detecting the problem structure.

5 Conclusion

In this paper, the design and usage of *Parampl* was presented, a parallel task submission extension for AMPL. Our experimental results prove that *Parampl* equips AMPL with a possibility of defining complex parallel algorithms solving optimization problems. It is able to take advantage of multiple processing units while computing the solutions. *Parampl* is very easy to deploy and use in AMPL programs and its implementation in Python programming language makes it platform independent.

References

1. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming, 2nd edn. Duxbury Press, Belmont (2002)
2. Boschetti, M., Maniezzo, V.: Benders decomposition, Lagrangean relaxation and metaheuristic design. J. Heuristics **15**, 283–312 (2009)
3. Dolan, E.D., Fourer, R., Goux, J.-P., Munson, T.S., Sarich, J.: Kestrel: an interface from optimization modeling systems to the NEOS server. INFORMS J. Comput. **20**, 525–538 (2008)
4. Colombo, M., Grothey, A., Hogg, J., Woodsend, K., Gondzio, J.: A structure-conveying modelling language for mathematical and stochastic programming. Math. Program. Comput. **1**(4), 223–247 (2009). doi:10.1007/s12532-009-0008-2
5. Fragnière, E., Gondzio, J., Sarkissian, R., Vial, J.-P.: Structure exploiting tool in algebraic modeling languages. Manage. Sci. **46**, 1145–1158 (2000)
6. Fourer, R., Gay, D.M.: Conveying problem structure from an algebraic modeling language to optimization algorithms. In: Laguna, M., González Velarde, J.L. (eds.) Computing Tools for Modeling, Optimization and Simulation. Interfaces in Computer Science and Operations Research, vol. 12, pp. 75–89. Kluwer, Boston (2000)
7. Buchanan, C.S., McKinnon, K.I.M., Skondras, G.K.: The recursive definition of stochastic linear programming problems within an algebraic modeling language. Ann. Oper. Res. **104**(1–4), 15–32 (2001)
8. Valente, C., Mitra, G., Sadki, M., Fourer, R.: Extending algebraic modelling languages for stochastic programming. INFORMS J. Comput. **21**(1), 107–122 (2009)
9. Fourer, R., Lopes, L.: StAMPL: a filtration-oriented modeling tool for multistage stochastic recourse problems. INFORMS J. Comput. **21**, 242–256 (2009)
10. Powell, M.J.D.: On the quadratic programming algorithm of Goldfarb and Idnani. Math. Program. Stud. **25**, 46–61 (1985)
11. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program. **106**(1), 25–57 (2006)

Prototyping Framework for Parallel Numerical Computations

Ondřej Meca, Stanislav Böhm(✉), Marek Běhálek, and Martin Šurkovský

Department of Computer Science, FEI VŠB Technical University of Ostrava, Ostrava, Czech Republic
{ondrej.meca,stanislav.bohm,marek.behalek,martin.surkovsky}@vsb.cz

Abstract. Our research is focused on the simplification of parallel programming for distributed memory systems. Our goal is to build a unifying framework for creating, debugging, profiling, and verifying parallel applications. The result of this effort is an open source tool Kaira. In this paper, we focus on prototyping of parallel applications. We have extended Kaira by the ability to generate parallel libraries. More precisely, we present a framework for fast prototyping of parallel numerical computations. We demonstrate our idea on a combination of parallel libraries generated by our tool Kaira and GNU Octave. Hence, a user can verify the idea in a short time, create a real running program and verify its performance and scalability.

Keywords: Prototyping · Parallel computing · Visual programming · Libraries

1 Introduction

Parallel computers are more and more available nowadays. A lot of people participate in developing parallel programs, but there are well-known difficulties of parallel programming. For example the user must learn how to use different specialized tools for profiling or debugging. Also, it usually takes more time to get a working parallel application that can be tested. Therefore, it can be difficult for many non-experts (even if they are experienced programmers of sequential applications) to make their programs run in parallel on a computer cluster.

The overall goal of our research is to reduce some complexity in parallel programming. In this paper, we focus on parallel application prototyping. More precisely, we present a framework for fast prototyping of parallel numerical computations. Hence, a user can verify his/her idea in a short time, create a real running program, and verify its performance and scalability. To address numerical computations, we demonstrate our idea on the combination of parallel libraries generated by our tool Kaira and GNU Octave[1] (Which we will now simply refer to as "Octave" in text). However, this approach can be easily generalized and

[1] http://www.gnu.org/software/octave/

generated libraries can be combined with other tools. We have chosen Octave because it represents a good example of a prototyping software, where users can easily experiment with their ideas.

2 Related Work

This section presents tools which have implemented ideas similar to Kaira and from which we have drawn inspiration. First, there were tools for visual programming of parallel applications. These tools were developed mainly in the 90s. As an example we can name *GRADE* [9], *CODE* [11] or HeNCE [3]. It is hard to evaluate these tools because they are no longer available or they run on no longer available hardware or operating systems. The semantics of our tools is similar to CODE but as far we know, CODE is not able to show a state of the application through the visual model. The same holds also for HeNCE that had similar features like CODE but expressiveness of its language was restricted. In GRADE, the application could be visually debugged but the visual language is based on different concept in comparison to our tool.

Despite that we are not aware of any such tool that is still actively developed or become widely accepted, we think that the visual approach to developing parallel applications is interesting and it deserve another chance. Parallel computers are more common and more accessible today; therefore, more scientific and engineering applications can profit from such hardware and not all of them require optimized handmade solutions. Additionally, we want to create a unified environment, where the same visual model is used not only during the development, but also to simplify various supportive activities.

A more successful approach to more abstract parallel programming is stream processing (StreamIt [12], FumeJava, Fastflow, etc.). FlumeJava [6] can be described as follows. It is a Java library for developing data-parallel pipelines. It offers lazy colle ction types and operations (map, filter, count, etc.), automatically generating and running a sequence of MapReduces only when the actual results are requested. MapReduce offers a good abstraction where many real parallel problems can be expressed. But data-flow is inherently limited by the target MapReduce framework – all data are processed uniformly in alternating and isolated map and reduce steps. From the perspective of these tools our tool offers a more low-level approach, less abstract programs. and more control over final applications. In our approach, we want to offer a more flexible environment where the user has more control to experiment with parallel algorithms.

Another approach is to introduce special constructions or annotations into widely used languages. As an example, we can name OpenMP[2], Unified Parallel C[3] [5] or Cilk++ [10]. The combination with standard languages makes these frameworks good prototyping tools, because a sequential program can be gradually parallelized. Many standard patterns, like parallelization of an independent for-cycle, can be also easily expressed in these tools. Our approach may

[2] http://openmp.org/wp/
[3] http://upc.lbl.gov/

need more work in the initial phase, because drawing a visual model is more demanding than setting up some annotations. But such model is useful for clear representation of the inner state of a running application, so it can speed up understanding of the application's behavior during testing, profiling, and other supportive activities.

If standard algorithms from a specific area are needed it is often the best solution to use some specialized libraries with tuned implementations. Considering numerical computations, there are specialized libraries for parallel numerical computations - for example libraries PETSc[4] and Trilinos[5]. However, when some special needs are required, it can be hard to adjust these libraries. They can be good prototyping tools and they can solve different problems, but if we want to experiment with different parallelization approaches, then they are usually not sufficient.

Use of such libraries is compatible with our approach. It is possible to combine their sequential parts with Kaira. For example, considering numerical computations, Kaira controls the overall data flow and Trilinos matrices are used for computations themselves.

3 Tool Kaira

This section serves as an overview for our tool Kaira; for more details see [1,2]. Our goal is to simplify the development of Message Passing Interface (MPI)[6] parallel applications and create an environment where all activities (prototyping, debugging, performance prediction, profiling, etc.) are unified under one concept.

The key aspect of our tool is the usage of visual models. In the first place, we have chosen visual models to obtain an easy and clear way to describe and expose parallel behavior of applications. The other reason is that a distributed state of an application can be shown through such visual model. The representation of an inner-state of distributed applications by a proper visual model can be more convenient than traditional ways like back-traces of processes and memory watches. Using this approach, we provide visual simulations where the user observes the behavior of the developed application. It can be used on incomplete applications from an early stage of development; therefore, it is a very useful feature for a prototyping tool. In a common way of development of MPI programs, it often takes a long time to get the developed application to a state where its behavior can be observed. We also use the same visual model for debugging and profiling. The user specifies through the visual model what to measure and Kaira generates a tracing version of the application recording its own runs. The record is presented back to the user through the original visual model of the application.

On the other hand, we do not want to create applications completely through visual programming. Sequential parts are written in the standard programming language (C++) and combined with the visual model that catches *parallel*

[4] http://www.mcs.anl.gov/petsc/

[5] http://trilinos.sandia.gov/

[6] http://www.mpi-forum.org/docs/docs.html

aspects and *communication*. We want to avoid huge unclear diagrams; therefore, we visually represent only what is considered as "hard" in parallel programming. Ordinary sequential codes are written in a textual language. Moreover, this design allows for easy integration of existing C++ codes and libraries.

It is important to mention that our tool is *not* an automatic parallelization tool. Kaira does not automatically detect parallelizable sections. The user has to explicitly define them, but they are defined in a high-level way and the tool derives implementation details.

Semantics of the Kaira visual programming language is based on Coloured Petri nets (CPNs) [8]. Petri nets are a formalism for description of distributed systems. They also provide well-established terminology, natural visual representation of models for their editing, and their simulations. The modeling tool *CPN Tools*[7] inspired us how to show visual models.

To demonstrate how our models work, let us consider a model in Fig. 1. It presents a problem where some jobs are distributed across computing nodes and results are sent back to process 0. When all these results arrive, they are written into a file. Circles (*places* in terminology of Petri nets) represent memory spaces. Boxes (*transitions*) represent actions. Arcs run from places to transitions (*input arcs*) or from transitions to places (*output arcs*). When a transition is executed it takes values (*tokens*) from places according to input arcs. When a computation in a transition is finished, then it produces new tokens to places according to output arcs. A computation described by this diagram runs on every process. Transferring tokens between processes are defined by the expression followed after character "@" in expressions on output arcs. A double border around a transition means that there is a C++ function inside. It is executed whenever the transition is fired. A double border around a place indicates an associated C++ function that defines the initial content of the place.

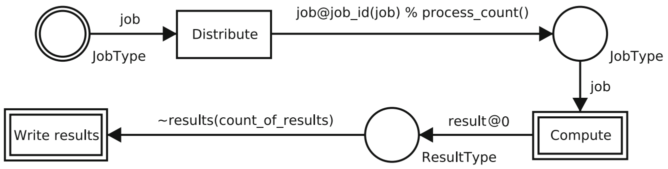


Fig. 1. The example of a model

4 Libraries

The infrastructure of libraries in Kaira is based on modules. A *module* is a model in Kaira enriched by an interface (depicted as a gray rectangle around the model). From a set of modules, Kaira generates a C++ library. As an example, consider the module in Fig. 2. It takes two input integers (x, y) and outputs a

[7] http://cpntools.org/

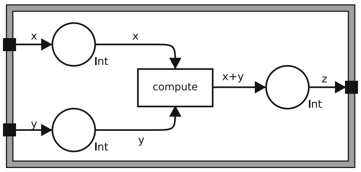


Fig. 2. The module *sum*

single integer (z). The example of a bigger module is presented in Sect. 5. When a library is generated from this module, we obtain the C++ library with the following interface:

```
void calib_init(int argc, char **argv);
void sum(int &x,int &y,int &z);
```

It is possible to use such library in any sequential C++ application. This application can be compiled and run through MPI infrastructure. The application will be executed in process 0 sequentially. When it calls a generated function, then the computation will be run across all MPI processes according to the structure of the module. When the function is finished (i.e. the module is finished) then the program continues again sequentially.

The library can be also generated in the *Remote Procedure Call* (RPC) mode. Kaira generates both server and client parts. The client side is a library that has the same interface as was described in the example above, but when a function is called it sends a request through a network to the server where a requested computation is executed.

4.1 Octave Libraries

Octave offers a possibility to create C++ modules (so called “oct-files”). It makes calling C++ functions accessible in the Octave environment. We use this infrastructure; Kaira is able to generate an oct-file that wraps our parallel libraries. Hence, the user is able to use modules smoothly in Octave in a similar way as in C++ applications. The important aspect of such integration is interoperability between data types. Kaira contains conversion functions for basic data types likes numbers or vectors. The user must provide conversion functions for own data types.

5 Case Study

As an example we have chosen a variant of the Finite Element Tearing and Interconnecting (FETI) domain decomposition method – Total-FETI [4][8]. Omitting other aspects like numerical scalability; parallelization of Total-FETI can be very

[8] The model and source codes used in this example are available on the website of our project http://verif.cs.vsb.cz/kaira.

straightforward. In [4], the basic idea is to decompose the domain into N subdomains. After the discretization, we get a block diagonal stiffness matrix (Eq. 2 in [4]), where matrices $\mathbf{K}_1 \dots \mathbf{K}_N$ are stiffness matrices for corresponding subdomains. Using such block diagonal stiffness matrix $\mathbf{K}$, we are usually able to divide computations and perform them in parallel (see the following equation).

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_1 & & \\ & \ddots & \\ & & \mathbf{K}_N \end{bmatrix}, \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \end{bmatrix}, \mathbf{Kx} = \begin{bmatrix} \mathbf{K}_1\mathbf{x}_1 \\ \vdots \\ \mathbf{K}_N\mathbf{x}_N \end{bmatrix}. \tag{1}$$

Such straightforward approach is far from being optimal. More advanced approaches were published in [7], where the authors focus on performance and usage of thousands of processors. In their solution, they use the library PETSc. Of course, such solution is much more time and resource demanding and relatively very complicated. On the other hand, Octave implementation that we start with roughly follows steps from paper [4] and it is relatively simple and readable. With such implementation, it is easy to explore different mathematical aspects or perform different experiments, but it is hard to address issues related to parallel programming.

More precisely, in the Octave API, there are external packages[9] for parallel/distributed computing. Package *general* contains two functions (`parcellfun` and `pararrayfun`) that evaluate functions using multiple processes. But it is restricted only to shared memory architectures. For distributed memory there are packages *openmpi_ext* and *parallel*. These packages are basic wrappers to MPI functions and simple sockets API. In both cases, they are quite low-level interfaces from a programmer's point of view. Tasks like debugging and profiling can be complicated considering development environments for Octave.

At the beginning, we had a working sequential implementation of Total-FETI for Octave. The most time consuming operation is a solution of linear system $\mathbf{Ky} = \mathbf{x}$. As was suggested in [4], Cholesky factorization of the stiffness matrix is used (`[L,ans,P]=chol(K,'lower')`). The following Octave code performs this time consuming computation:

```
function y=Kplus_aux(L, P, x)
    Lt=L';
    Pt=P';
    y=P*(Lt\(L\(Pt*x)));
end
```

Total-FETI iteratively computes the result and thus uses `Kplus_aux` several times. We want to parallelize this operation using Kaira and explore its properties. Module `Kplus_par` from the Fig. 3 defines the parallel computations. In this model we use types *Matrix* and *SparseMatrix* which are native types offered by the Octave C++ interface.

Matrices $\mathbf{L}$ and $\mathbf{P}$ are a block diagonal. First, they are (along with vector $\mathbf{x}$) divided according to their block diagonal structure (transition *Divide*). By

[9] All mentioned packages are available at http://octave.sourceforge.net

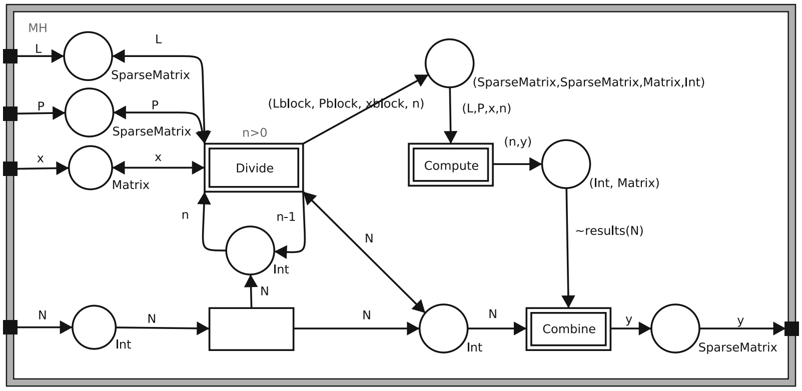


Fig. 3. Module *Kplus_par*

this action we obtain N smaller tasks. These tasks are processed by transition *Compute*. A source code in the transition *Compute* performs the same computation like the original Octave function `Kplus_aux`, but using a single block that represents one sub-domain. The following source code is stored in the transition *Compute*. The user needs to write only the three lines in the body of the function. The rest is a template generated by the tool.

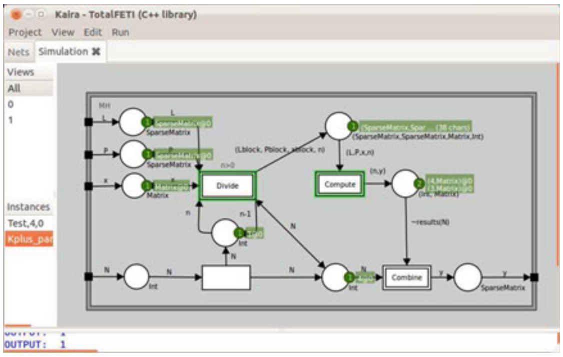
```
struct Vars {
    Matrix x; Matrix y;
    SparseMatrix L; SparseMatrix P;
    int n;
};
void transition_fn(CaContext &ctx, Vars &var) {
    SparseMatrix Lt=var.L.transpose();
    SparseMatrix Pt=var.P.transpose();
    var.y = var.P*(Lt.solve(var.L.solve(Pt*var.x)));}
}
```

When all partial results are produced, transition *Combine* is fired and the resulting vector is composed. After that, the module is terminated and the data are transmitted back to the Octave. In the original source code for Octave, the only change is the call of the generated function (instead of the original sequential one). It has one additional parameter N indicating the number of sub-domains.

The model shown in Fig. 3 represents a solution for a shared memory system. If the resulting application is started in a configuration with multiple threads, then it is performed in parallel. An extension of this net for usage with distributed memory is easy. We just need to modify the existing arc inscription: *(Lblock, Pblock, xblock, n)* to: *(Lblock, Pblock, xblock, n)@n* and *(n, y)* to *(n, y)@0*. It causes that blocks will be assigned to MPI processes according to their positions and results are sent back to process 0.

5.1 Experiments and Results

Now with the existing model, a user is able to use various features of Kaira. For example it is possible to perform simulations of module executions independently on Octave code (Fig. 4), where the user manually controls the simulation. Additionally, it is possible to run the application in the tracing mode where the execution of a module is recorded. Such recorder execution can be replayed using the original model or some performance statistics can be obtained (like execution times for each transition, etc).



```
OUTPUT:  1
```

Fig. 4. A screenshot of a simulation

To test the module's performance, we solve a displacement of a 1D string that is fixed on both ends. We prepare a stiffness matrix where each sub-domain has 500000 discretization steps and we use 30 sub-domains. The measurement was performed on a computer with 8 processors AMD Opteron/2500 (having a total of 32 cores). A computation of original `Kplus_aux` takes in average 21.79 s in the pure Octave solution. We measured runs of the library generated from our module `Kplus_par`. The test was performed in RPC mode, where both client and server run on the same computer. The measured times for multithreading and MPI backends are listed in Table 1.

These results are consistent with reasonable expectations. They show that there is a communication cost related to the RPC mode (difference between running times for multithreading with and without RPC). This cost is fixed due

Table 1. Running times (in seconds) of *Kplus_par* using threads and MPI

Nodes	1	2	4	8	16	32
Threads + RPC	25.30	19.87	10.32	8.02	7.70	7.76
MPI + RPC	25.19	20.88	16.89	15.36	16.55	16.39
Threads (no RPC)	21.71	11.05	6.51	4.35	4.06	3.93

to the fixed problem size. It also presents a bottleneck for further performance improvements. For multithreading we reached this bottleneck around 16 cores. At this point, a time to distribute matrices is much bigger than a time to perform the computation itself. Usage of MPI introduces additional communication overhead, because data are distributed between nodes using MPI functions.

The execution of the sequential version (using only one core) of the whole computation takes approximately 320 s while the function `Kplus_aux` was used 5 times. For real problems, the number of iterations can be different (usually bigger) and when the stiffness matrices contain more non-zero elements, the function `Kplus_aux` will be more time consuming. We present these results mainly to prove, that we are able to get a working parallel solution with reasonable performance and even if the obtained solution may not be the most optimal, we were able to develop it fast.

To keep the presented solution simple, the stiffness matrix $\mathbf{K}$ is divided and distributed for every computation. To improve the performance further, we can store these blocks in computing nodes and use them several times, while they do not change during the computation. Mentioned matrices are sparse, but they are usually huge. Their size is based on the number of primary variables and there can be millions of primary variables in real problems. So for real experiments the memory consumption often becomes an issue (usually even before the performance). In fact, we do not need to compose the whole stiffness matrix. When we divide its blocks between computing nodes, we can handle a problem originally too large for a single computer. This can be an even bigger advantage than just the performance improvement.

6 Conclusion

The paper presents our tool Kaira and its ability to generate parallel libraries. Also it demonstrates their usage in a combination with Octave. Such combination allows rapid prototyping of parallel numerical computations. On a real example (TotalFETI method), we demonstrated that it takes only a few lines of C++ code and a relatively simple model based on CPNs to get a working parallel application with reasonable performance. Further, this model can be easily extended and we are use the same visual model for debugging and profiling. Thus, an inexperienced user does not have to learn additional tools.

Kaira is still being actively developed. We are trying to improve the process of gathering information from the model to provide functions like performance prediction or verification. From the perspective of this paper, we are also focused on additional simplifications for binding Octave types and allow for preserving data on computing nodes during consecutive computations. We also want to extend our approach to Matlab.

Acknowledgments. The work is partially supported by: GAČR P202/11/0340, the European Regional Development Fund in the IT4Innovations Center of Excellence project (CZ.1.05/1.1.00/02.0070) and Grant of SGS No. SP2013/145, VŠB - Technical University of Ostrava, Czech Republic.

References

1. Böhm, S., Běhálek, M.: Generating parallel applications from models based on petri nets. Adv. Electr. Electron. Eng. **10**(1), 28–34 (2012)
2. Böhm, S., Běhálek, M.: Usage of Petri nets for high performance computing. In: Proceedings of the 1st ACM SIGPLAN Workshop on Functional High-Performance Computing, FHPC '12, pp. 37–48. ACM, New York (2012) http://doi.acm.org/10.1145/2364474.2364481
3. Browne, J.C., Dongarra, J., Hyder, S.I., Moore, K., Newton, P.: Visual programming and parallel computing. Technical report, Knoxville, TN, USA (1994)
4. Dostál, Z., Horák, D., Kučera, R.: Total FETI-an easier implementable variant of the FETI method for numerical solution of elliptic PDE. Commun. Numer. Meth. Eng. **22**(12), 1155–1162 (2006). http://dx.doi.org/10.1002/cnm.881
5. El-Ghazawi, T., Smith, L.: UPC: unified parallel C. In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, SC '06. ACM, New York. http://doi.acm.org/10.1145/1188455.1188483 (2006)
6. Gordon, M.I., Thies, W., Amarasinghe, S.: Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. SIGPLAN Not. **41**(11), 151–162 (2006)
7. Horák, D., Dostál, Z.: Parallelization of the total-FETI-1 algorithm for contact problems using PETSc. In: Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering. Civil-Comp Press, Stirlingshire (2011)
8. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer, Heidelberg (2009)
9. Kacsuk, P., Cunha, J., Dózsa, G., Lourenco, J., Fadgyas, T., Antao, T.: A graphical development and debugging environment for parallel programs. Parallel Comput. **22**(13), 1699–1701 (1997). http://www.sciencedirect.com/science/article/pii/S0167819196000750 (distributed and parellel systems: Environments and tools)
10. Leiserson, C.: The Cilk++ concurrency platform. J. Supercomput. **51**(3), 244–257 (2010). http://dx.doi.org/10.1007/s11227-010-0405-3
11. Newton, P., Browne, J.C.: The code 2.0 graphical parallel programming language. In: Proceedings of the 6th International Conference on Supercomputing, ICS '92, pp. 167–177. ACM, New York. http://doi.acm.org/10.1145/143369.143405 (1992)
12. Thies, W., Amarasinghe, S.: An empirical characterization of stream programs and its implications for language and compiler design. In: Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques, PACT '10, pp. 365–376. ACM, New York (2010)

Algorithms for In-Place Matrix Transposition

Fred G. Gustavson[1,2] and David W. Walker[3(✉)]

[1] IBM T. J. Watson Research Center (emeritus), Yorktown Heights, USA
[2] Umeå University, Umeå, Sweden
[3] School of Computer Science and Informatics, Cardiff University, Cardiff, UK
WalkerDW@cardiff.ac.uk

Abstract. This paper presents an implementation of an in-place swap-based algorithm for transposing rectangular matrices, and a proof of correctness is also sketched. The implementation is based on an algorithm described by Tretyakov and Tyrtyshnikov [4], but we have introduced a number of variations. In particular, we show how the original algorithm can be modified to require constant additional memory. We also identify opportunities for exploiting parallelism.

Keywords: Matrix · Transpose · Permutations · Performance · Parallelism

1 Introduction

The transposition of rectangular matrices appears in linear algebra computations, and is important in a number of areas. For example, a common approach to finding the discrete Fourier transform of a three-dimensional array is to perform a 1D transform with respect to each dimension in turn and to transpose the data with respect to two of the dimensions between each of the transformation stages. This is done to ensure that the data being transformed in 1D is always contiguous; this improves performance by using cache efficiently, and it facilitates parallelization.

Let A be a contiguous matrix of size $n \times m$ elements. Given extra memory of size nm matrix elements, an out-of-place matrix transposition can be performed by simple memory-to-memory copies, but only by using large stride operations. However, for sufficiently large matrices this is often not an efficient use of memory. In such cases an in-place transposition algorithm is preferable because it evaluates the transpose using less memory. In Sect. 4 the in-place swap-based transposition algorithm of Tretyakov and Tyrtyshnikov [4] is described. This uses $\min(n, m)$ extra storage. Henceforth, for brevity, this is called the TT algorithm. Specifically, we give variations of the TT algorithm, and discuss their characteristics; i.e., the amount of extra memory required, the efficiency of memory access, the potential for parallelization, and their costs. The novel contributions of this paper are as follows:

1. To the best of the authors' knowledge theirs is the first implementation of the TT algorithm, and this paper presents the results of a practical performance study of this algorithm for the first time.
2. The paper shows that the additional $O(m)$ memory required by the TT algorithm to transpose an $n \times m$ matrix can be reduced to $O(1)$.
3. The paper highlights opportunities for parallelism, and demonstrates that a divide-and-conquer approach can reduce the number of swaps required in a key component of the TT algorithm.
4. The paper compares the measured runtimes of the TT algorithm and other in-place transposition algorithms, and shows that these can execute faster than the TT algorithm, even though they have a higher swap count. This is attributed to more irregular memory access patterns in the TT algorithm.

The remainder of this paper is organised as follows. Section 2 introduces some notation and terminology, and Sect. 3 outlines the use of cycle-following algorithms for in-place matrix transposition. This is followed in Sect. 4 by a description of the swap-based TT algorithm [4]. Variants of the TT algorithm are discussed in Sect. 5, with particular attention to memory requirements and the potential for exploiting parallelism. The paper's main conclusions are then presented in Sect. 6.

2 Notation and Terminology

Computer memory may be viewed as a linear sequence of uniformly-spaced storage locations. A set of items $a_0, a_1, \ldots, a_{r-1}$ is said to be stored contiguously if item a_i is stored starting at the next storage location following the end of item a_{i-1}, for $i = 1, \ldots, r-1$. The standard storage scheme for a matrix of size $n \times m$ can be specified by a mapping from row i and column j to a offset number k. Thus, for a column-major order (CMO) matrix the mapping is $k = j \times n + i$, while for a row-major order (RMO) matrix it is $k = i \times m + j$.

It should be noted that transposing a CMO matrix is equivalent to transforming its storage scheme from CMO to RMO. Similarly, transposing a RMO matrix is equivalent to transforming its storage scheme from RMO to CMO.

3 Matrix Transposition Based on Cycle Following

There is a vast literature on in-place matrix transposition and the more general topic of in-place permutation. A detailed discussion can be found in the recent paper by Gustavson, Karlsson and Kagstrom [2], which also contains a full section on related work on in-place transposition.

For a CMO matrix A of size $n \times m$, a_{ij} stored at index $k = jn + i$ is stored at index $\hat{k} = im + j$ of A^{T}, which is also stored in CMO format. So the basic idea behind cycle-following algorithms is that matrix transposition can be viewed as a permutation $\hat{k} = P(k)$ of the integers $0, 1, \ldots, q$, where $q = mn - 1$.

The elements $k = 0$ and $k = q$ are always invariant under P, and for $0 < k < q$ we have:

$$P(k) = km \bmod q.$$

If the permutation P is applied repeatedly eventually one returns to the starting index, i.e., there is some integer c such that $k = P^c(k)$. The values $k, P(k), \ldots, P^{c-1}(k)$ define a cycle of the permutation, and c is called the cycle length. In general, the transposition of a matrix can be viewed as the union of several disjoint cycles. For example, consider $n = 5$ and $m = 3$, where $q = mn-1 = 14$. The corresponding transposition permutation contains the following fives cycles:

$$(0)\ (3, 9, 13, 11, 5, 1)\ (7)\ (10, 2, 6, 4, 12, 8)\ (14)$$

Elements $k = 0, 7, 14$ are cycles of length 1 and require no data movement. The first cycle of length 6 requires the following data movement. First make a copy of $A[1]$, the element of A at index 1. Then move $A[5] \rightarrow A[1]$, $A[11] \rightarrow A[5]$, $A[13] \rightarrow A[11]$, $A[9] \rightarrow A[13]$, and $A[3] \rightarrow A[9]$. Finally move the copy of $A[1]$ to $A[3]$. By treating the cycle $(10, 2, 6, 4, 12, 8)$ in the same way the transposition is completed.

Cycle-following algorithms for in-place transposition consist of a first phase in which exactly one member of each cycle is identified. This is referred to as the cycle set leader. In the second phase P^{-1} is applied to each cycle leader and the appropriate elements are moved round the cycle.

4 Matrix Transposition Based on Swaps

The TT paper [4] transposes in-place a $n \times m$ matrix A with $n \geq m$, where A is stored in CMO. It has two main phases. In the first phase A is partitioned in-place over its rows into a set of $k+1$ contiguous sub-matrices $A_i, i = 0, \ldots, k$, each of which is stored contiguously in CMO. This is done by means of k applications of the unshuffle operation, described below. In the second phase these $k+1$ sub-matrices A_i are transposed. This completes the transposition of A. This approach is readily adapted to handle the transposition of matrices stored in row-major order and/or the case $n < m$. The transposition of each sub-matrix in-place can be done independently, so either all the sub-matrices can be generated and then transposed, or each can be transposed as soon as it is generated before generating the next sub-matrix. In our implementation we have chosen the former approach for clarity of exposition as well as to expose parallelism.

As discussed in Sects. 4.1 and 4.2, the two phases of the TT algorithm for transposing an $n \times m$ matrix may be described in terms of the following swap-based operations:

1. Unshuffle vectors of contiguous values. Consider the following contiguous sequence of columns of the $n \times m$ matrix A: $a_1 b_1 a_2 b_2 \ldots a_m b_m$ where each a_i is itself a contiguous vector of ℓ_a elements, and each b_i is a contiguous vector

of ℓ_b elements, with $\ell_a \geq \ell_b$ and initially $\ell_a + \ell_b = n$. Then the unshuffle operation corresponds to the following transformation:

$$a_1 b_1 a_2 b_2 \ldots a_m b_m \rightarrow a_1 a_2 \ldots a_m b_1 b_2 \ldots b_m \quad (1)$$

The number of swaps for this operation does not exceed $2m\ell_a + 3m\ell_b + m^2/2$ [4]. Also, m elements of additional storage are required.

2. Shuffle vectors of contiguous values. The shuffle operation performs the operations of unshuffle in reverse order so the operation bound and additional storage value are identical. Shuffle performs the transformation

$$a_1 a_2 \ldots a_m b_1 b_2 \ldots b_m \rightarrow a_1 b_1 a_2 b_2 \ldots a_m b_m. \quad (2)$$

3. The vector transposition of a $pm^q \times m$ matrix, where $0 < p < m$, $q > 0$, and the vector length is p. This is equivalent to the transposition of an $m^q \times m$ matrix in which each element is a column vector of length p. The authors of [4] report that the number of swaps for this operation does not exceed pm^{q+1}, and an additional q items of storage are required.
4. The in-place transposition of a square matrix. For a $k \times k$ matrix this requires $k(k-1)/2$ swaps and additional storage for one item.
5. The out-of-place transposition of a non-square matrix having fewer than m entries. This operation requires an index calculation followed by a copy for each matrix element, and uses an additional m items of storage.
6. A permutation. Given a vector c of vector elements and a permutation vector v, i.e., a vector containing the integers 1 to n in some order, then the permutation induced by v causes the vector at $c(i)$ to be stored at index $v(i)$ of c. For example, if $c = (1, 2, 3, 4, 5, 6)$ and $v = (2, 4, 1, 6, 5, 3)$, then applying the permutation v to c gives (3,1,6,2,5,4).

4.1 The Partition Phase

The partition phase proceeds as follows. First n is expressed as a radix-m number consisting of $k+1$ digits: $n = n_k m^k + n_{k-1} m^{k-1} + \cdots + n_1 m + n_0$ where $0 \leq n_i < m$, and $n_k \neq 0$. The matrix A is written as $A = \{A_k, A_{k-1}, \ldots, A_1, A_0\}$, where A_k is an $n_k m^k \times m$ matrix consisting of the first $n_k m^k$ rows of A, A_{k-1} is an $n_{k-1} m^{k-1} \times m$ matrix consisting of the next $n_{k-1} m^{k-1}$ rows of A, and so on, with A_0 being an $n_0 \times m$ matrix consisting of the last n_0 rows of A. Thus, in general A_i is an $n_i m^i \times m$ matrix, whose elements are stored in CMO.

The transformations carried out in each phase of the transpose algorithm are illustrated in Fig. 1 for a 26×4 matrix. Since $26 = 1 \times 4^2 + 2 \times 4 + 2$ the original matrix is partitioned into $k+1 = 3$ sub-matrices of size 16×4, 8×4, and 2×4. As may be seen in the middle matrix shown in Fig. 1, each of the sub-matrices is stored contiguously in column-major order.

A high-level description of the algorithm of [4] is shown in Algorithm 1, in which the IPSB_Partition routine overwrites the input matrix, A, with the sub-matrices A_i for $i = 0, 1, \ldots, k$. As discussed above, the sub-matrices generated have a special form: A_i has size $n_i m^q \times m$, where $n_i < m$ and $q \geq 0$. The routine

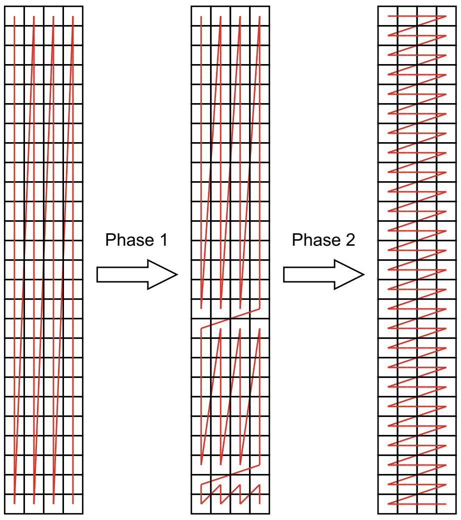


Fig. 1. Transformations carried out in each phase of the swap-based matrix transpose algorithm illustrated with a 26×4 matrix. Each square represents one matrix element, and the continuous line between cell centers shows the order in which they are stored, starting in the top left corner. Phase 1 is carried out in $k = 2$ stages: $(26) \rightarrow (16, 10) \rightarrow (16, 8, 2)$.

Algorithm 1. IPSB_Transpose: In-Place Swap-Based Matrix Transpose

```
Input: Matrix $A$ of size $n \times m$, and work-space $w$ of size $iw$.
Output: The number of swaps performed. The matrix $A$ is overwritten by $A^{\mathrm{T}}$.
$k$ = NumberOfDigits($n$,$m$) - 1;
$nswaps$ = IPSB_Partition($A$, $n$, $m$, $w$, $iw$);
foreach sub-matrix $A_i$, $i$ = 0,1,...,$k$ do
    $ni$ = Digit($i$,$n$,$m$); $q$ = Power($m$,$i$);
    $nswaps$ = $nswaps$ + IPSB_PowerTranspose($A_i$,$ni$,$q$,$m$,$w$,$iw$);
end
return [$nswaps$]
```

IPSB_PowerTranspose uses swap-based methods to transpose matrices of this type. Algorithm 1 also contains some auxiliary routines: NumberOfDigits(n,m) returns the number of digits in the base-m representation of n; Digit(i,n,m) returns digit i in such a representation; and Power(m,i) returns m^i.

The type of partitioning described above can be performed using k applications of the unshuffle operation. The matrix is initially stored contiguously in CMO as follows: $a_1^k \dots a_1^0 a_2^k \dots a_2^0 \dots a_m^k \dots a_m^0$ where a_j^i is column j of sub-matrix A_i. In the first application of the unshuffle operation $\ell_a = n_k m^k$ and $\ell_b = n - \ell_a$, and the columns of the first sub-matrix, A_k, are gathered together and made contiguous as follows:

$$a_1^k \dots a_1^0 a_2^k \dots a_2^0 \dots a_m^k \dots a_m^0 \rightarrow a_1^k a_2^k \dots a_m^k a_1^{k-1} \dots a_1^0 a_2^{k-1} \dots a_2^0 \dots a_m^{k-1} \dots a_m^0.$$

In the second application of the unshuffle operation the columns of A_{k-1} are gathered together and made contiguous, and in general in the ith application of the unshuffle operation the columns of A_{k+1-i} are gathered and made contiguous. Thus, after k applications of the unshuffle operation the columns of the

Algorithm 2. IPSB_Partition: In-Place Swap-Based Partitioning of Matrix

```
Input: Matrix $A$ of size $n \times m$, and work-space $w$ of size $iw$.
Output: The number of swaps performed. The matrix $A$ is overwritten by
        sub-matrices $A_i$, $i = 0, 1, \ldots, nd - 1$, where $nd$ is the number of digits in $n$
        to the base $m$.
$nd$ = NumberOfDigits($n$,$m$); $mpow$ = Power($m$,$nd$-1); $p = 0$; $nt = n$;
while $mpow > 0$ do
    $ni = nt/mpow$; $la = ni * mpow$; $lb = nt$-$la$;
    $nswaps = nswaps$ + Unshuffle($A[p]$,$la$,$lb$,$w$,$iw$);
    $p = p + m * la$; $nt = lb$; $mpow = mpow/m$;
end
return [$nswaps$]
```

sub-matrices are ordered as follows: $a_1^k \ldots a_m^k a_1^{k-1} \ldots a_m^{k-1} \ldots a_1^0 \ldots a_m^0$ which is the required ordering. The IPSB_Partition routine is presented in Algorithm 2.

4.2 The Transpose Phase

In the transpose phase each of the now contiguous $k + 1$ sub-matrices A_i is transposed. Each A_i has size $n_i m^i \times m$ and it is transposed as follows:

1. Perform a vector transpose with vector size n_i. As shown in Fig. 2, this results in the matrix being stored as m^i matrices $B_j, j = 1, \ldots, m^i$, each of size $n_i \times m$.
2. Each B_j is transposed by dividing its columns into $g_i = m/n_i$ square matrices of order n_i and one $n_i \times r_i$ matrix R_j, where $r_i = m \bmod n_i$. Thus R_j contains the last r_i columns of B_j. Each of the square $n_i \times n_i$ sub-matrices of each B_j is transposed in-place, and each R_j matrix is also transposed in place. The TT paper [4] gives no specific guidance on how to transpose the R_j, but our implementation uses a recursive call to our general transpose algorithm. The recursion terminates when the matrix to be transposed has fewer than m elements, when an out-of-place algorithm is used to do the transposition.
3. At this stage of the algorithm the storage for each B_j contains:

   $$w_1^1 \ldots w_{n_i}^1 \ldots w_1^{g_i} \ldots w_{n_i}^{g_i} \rho_1 \ldots \rho_{n_i}$$

   where w_q^p represents the qth row of the pth square matrix, and ρ_q the qth column of matrix R_j^{T}. Note that $\ell(w_q^p) = n_i$ and $\ell(\rho_q) = r_i$. Ignoring for now the ρ_q, the following permutation is performed:

   $$w_1^1 \ldots w_{n_i}^1 \ldots w_1^{g_i} \ldots w_{n_i}^{g_i} \rightarrow w_1^1 w_1^2 \ldots w_1^{g_i} \ldots w_{n_i}^1 w_{n_i}^2 \ldots w_{n_i}^{g_i}.$$

   In [4] this requires an extra array of size $n_i g_i$, which does not exceed m.
4. The shuffle operation is used to complete the transposition of each B_j by performing the following transformation:

   $$w_1^1 w_1^2 \ldots w_1^{g_i} \ldots w_{n_i}^1 w_{n_i}^2 \ldots w_{n_i}^{g_i} \rho_1 \ldots \rho_{n_i} \rightarrow w_1^1 w_1^2 \ldots w_1^{g_i} \rho_1 \ldots w_{n_i}^1 w_{n_i}^2 \ldots w_{n_i}^{g_i} \rho_{n_i}.$$

These four operations that constitute the transpose phase are illustrated in Fig. 2 for a particular sub-matrix A_i with $q = 1$, $n_1 = 3$, and $m = 8$.

4.3 The Exchange Operation

Algorithm 1 gives a high level view of the TT [4] algorithm of Tretyakov and Tyrtyshnikov, and further details are given in Algorithms 2 and 3. However, the exchange operation forms an important building block of the shuffle and unshuffle operations. It takes two contiguous vectors of length p and q that are contiguous themselves and reverses their order in-place:

$$a_1 \dots a_p b_1 \dots b_q \rightarrow b_1 \dots b_q a_1 \dots a_p \tag{3}$$

The exchange operation requires *only* one extra element, regardless of the values of p and q. We show in [3] that the number of swaps performed is

$$p + q - \gcd(p, q). \tag{4}$$

If $p \geq q$, the items $b_1 \dots b_q$ are exchanged with $a_1 \dots a_q$ to give $b_1 \dots b_q a_{q+1} \dots a_p$ $a_1 \dots a_q$. However, if $p < q$, then items $a_1 \dots a_p$ are exchanged with $b_{q-p+1} \dots b_q$ to give $b_{q-p+1} \dots b_q b_1 \dots b_{q-p} a_1 \dots a_p$.

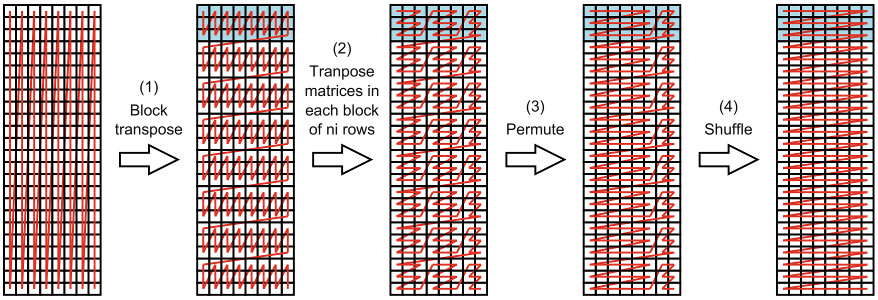


Fig. 2. Transformations carried out in the second phase of the swap-based matrix transpose algorithm for one sub-matrix, illustrated for the case $q = 1$, $n_1 = 3$, and $m = 8$. Each square represents one matrix element, and the continuous line between cell centers shows the order in which they are stored.

Assuming, without loss of generality, that $p \geq q$, then after the first exchange the $b_1 \dots b_q$ are in the correct location, and the order of $a_{q+1} \dots a_p$ and $a_1 \dots a_q$ needs to be reversed, which can be done by repeating the process.

4.4 The Shuffle and Unshuffle Operations

The exchange operation is repeatedly used in the shuffle and unshuffle operations. The shuffle operation is the same as the unshuffle operation, but performs its steps in reverse order, so only the unshuffle operation, which performs the reordering shown in Eq. 1, will be described here. In Eq. 1 each a_i and b_i is a vector of length ℓ_a and ℓ_b, respectively.

Algorithm 3. IPSB_PowerTranspose: In-Place Transpose of $pm^q \times m$ Matrix

```
Input: Matrix $A$ of size $pm^q \times m$, and work-space $w$ of size $iw$.
Output: The number of swaps performed. The matrix $A$ is overwritten by $A^T$.
$nswaps = nswaps$ + IPSB_VectorTranspose($A$,$p$,$q$,$m$,$w$,$iw$); $ni = p$;
foreach $ni \times m$ block matrix $B_j$, $j = 1, 2, \ldots, m^q$ do
    $bptr = (j-1)*ni*m$; $gi = m/ni$; $sbptr = bptr$; $ri$ = Mod($m$,$ni$);
    foreach $ni \times ni$ sub-block of current block, $k = 0, 1, \ldots, gi-1$ do
        $nswaps = nswaps$ + IPSB_SquareTranspose($A(sbptr)$,$ni$);
        $sbptr = sbptr+ni*ni$;
    end
    $nswaps = nswaps$ + IPSB_Transpose($A(sbptr)$,$ri$,$ni$,$w$,$iw$); ! see Algorithm 1;
    $nswaps = nswaps$ + IPSB_Permute($A(bptr)$,$ni$,$m$,$w$,$iw$); ! see 3 of Sect. 4.2;
    $nswaps = nswaps$ + IPSB_Shuffle($A(bptr)$,$gi*ni$,$ri$,$ni$,$w$,$iw$); ! see 4 of Sect. 4.2;
end
return [$nswaps$]
```

The algorithm for the unshuffle operation proceeds in m steps. At the start of step j, for $j = 1, \ldots, m-2$, the first j of the a vectors are assumed to have been placed in their correct positions through a four-step process, described below, that may result in a_{j+1} being split into two parts, $\bar{a}_{j+1}$ and $\tilde{a}_{j+1}$, such that, when together again, $a_{j+1} = \bar{a}_{j+1}\tilde{a}_{j+1}$. Thus, in general, at the start of step j the ordering is:

$$a_1 \ldots a_j \bar{a}_{j+1} b_{v_1} \ldots b_{v_j} \tilde{a}_{j+1} b_{v_{j+1}} a_{j+2} \ldots b_{v_m} \tag{5}$$

where $v_1, \ldots, v_m$ is a permutation of the indices $1, \ldots, m$;.

In step j item a_{j+1} is moved to its correct location, so that at the end of step j (and the beginning of step $j+1$) we have

$$a_1 \ldots a_{j+1} \bar{a}_{j+2} b_{v_1} \ldots b_{v_{j+1}} \tilde{a}_{j+2} b_{v_{j+2}} a_{j+3} \ldots b_{v_m} \tag{6}$$

The proof of correctness of the unshuffle operation is to show the induction hypothesis of Eq. 5 is true. TT in [4] show this by breaking down step j into the following four substeps that transform the ordering from that shown in Eq. 5 to that shown in Eq. 6.

1. Use swaps to move $\tilde{a}_{j+1}$ to the correct position immediately after $\bar{a}_{j+1}$, thereby reforming a_{j+1}. This may cause a split in one of the b vectors to give: $a_1 \ldots a_{j+1} \tilde{b}_{v_r} b_{v_{r+1}} \ldots b_{v_j} b_{v_1} \ldots b_{v_{r-1}} \bar{b}_{v_r} b_{v_{j+1}} a_{j+2} \ldots b_{v_m}$.
2. Possibly split vector a_{j+2} by swapping the positions of $\tilde{b}_{v_r}$ and $\bar{a}_{j+2}$ to give: $a_1 \ldots a_{j+1} \bar{a}_{j+2} b_{v_{r+1}} \ldots b_{v_j} b_{v_1} \ldots b_{v_{r-1}} \bar{b}_{v_r} b_{v_{j+1}} \tilde{b}_{v_r} \tilde{a}_{j+2} \ldots b_{v_m}$.
3. Use the exchange operation on $b_{v_{j+1}} \tilde{b}_{v_r}$ to get $\tilde{b}_{v_r} b_{v_{j+1}}$ which then gives $a_1 \ldots a_{j+1} \bar{a}_{j+2} b_{v_{r+1}} \ldots b_{v_j} b_{v_1} \ldots b_{v_r} b_{v_{j+1}} \tilde{a}_{j+2} \ldots b_{v_m}$.
4. The permutation vector v has possibly changed in going from Eq. 5 to Eq. 6. So, record its change in this step: $v_1 \ldots v_m \rightarrow v_{r+1} \ldots v_j v_1 \ldots v_r v_{j+1} \ldots v_m$. This is done by applying an exchange operation to v in which the first set

of items is $v_1 \ldots v_r$ and the second set of items is $v_{r+1} \ldots v_j$. After this the ordering is that given in Eq. 6, in readiness for the next substep.

After step $m-2$ we have, according to Eq. 6,

$$a_1 \ldots a_{m-1} \bar{a}_m b_{v_1} \ldots b_{v_{m-1}} \tilde{a}_m b_{v_m}. \tag{7}$$

In step $m-1$ the exchange operation is then used to exchange to $b_{v_1} \ldots b_{v_{m-1}}$ and $\tilde{a}_m$ to give

$$a_1 \ldots a_m b_{v_1} \ldots b_{v_m} \tag{8}$$

and in step m a vector permute inverse operation is then applied using the permutation vector v to obtain the required ordering: $a_1 \ldots a_m b_1 \ldots b_m$.

5 Variations and Alternatives to the TT Algorithm

5.1 A Divide-and-Conquer Version of the Shuffle and Unshuffle Operations

The shuffle and unshuffle operations described in Sect. 4.4 can be implemented using a readily parallelisable divide-and-conquer (DAC) approach that uses exchange operations. To illustrate this consider a case with $m = 8$, and divide the a and b vectors into groups as follows: $(a_1b_1a_2b_2)(a_3b_3a_4b_4)(a_5b_5a_6b_6)(a_7b_7a_8b_8)$. Now exchange the first b vector with the second a vector in each group to give: $(a_1a_2b_1b_2)(a_3a_4b_3b_4)(a_5a_6b_5b_6)(a_7a_8b_7b_8)$. Next group as follows:$(a_1a_2b_1b_2a_3a_4 b_3b_4)(a_5a_6b_5b_6a_7a_8b_7b_8)$, and exchange the first pair of b vectors with the second pair of a vectors in each group to give: $(a_1a_2a_3a_4b_1b_2b_3b_4)(a_5a_6a_7a_8b_5b_6b_7b_8)$. The final step is to treat all the vectors as a single group and exchange the first set of four b vectors with the second set of four a vectors to obtain the required ordering: $(a_1a_2a_3a_4a_5a_6a_7a_8b_1b_2b_3b_4b_5b_6b_7b_8)$.

We found it not possible to derive a closed-form expression for the number of swaps in the algorithms given by TT for the shuffle and unshuffle operations. However, for the DAC-based shuffle and unshuffle algorithms with n_p-way parallelism, where m and n_p are powers of 2, the number of swaps is:

$$\left[m\left(1-\frac{1}{n_p}\right)+\left(\frac{m}{2n_p}\right)\log_2\left(\frac{m}{n_p}\right)\right](\ell_a+\ell_b-g). \tag{9}$$

where g is the greatest common divisor of ℓ_a and ℓ_b. If $n_p = 1$ the number of swaps for the sequential case is obtained, and an upper bound on the speed-up can be calculated as:

$$S(m, n_p) = \frac{n_p \log_2 m}{2(n_p-1)+\log_2\left(\frac{m}{n_p}\right)}. \tag{10}$$

This is an upper bound because all sources of overhead arising from the parallel execution (synchronization, data movement) have been ignored. The maximum amount of parallelism that can be extracted from the DAC-based algorithm corresponds to the case $n_p = m/2$ for which the upper bound on the speed-up reduces to:

$$\frac{\log_2 m}{2\left(1 - \frac{1}{m}\right)} \tag{11}$$

The lefthand plot in Fig. 3 compares the number of swaps measured for the unshuffle operation for the algorithm given by TT with the number for the DAC-based algorithm described above. The dashed line shows results for the case in which maximum parallelism is exploited for each value of m, i.e., when $n_p = m/2$. Even for the sequential DAC-based algorithm for large problems the swap count is lower than for the TT case. In general, we have found that the relative number of swaps for the TT algorithm and our DAC-based algorithm depends of the values of ℓ_a, ℓ_b, and m, and for smaller values of m the number of swaps may be larger in the DAC case. However, the DAC-based algorithm avoids splitting any a or b vectors, and avoids permuting the b vectors. For the parallel DAC-based algorithm with $n_p = 4$ and 16 the swap count is always lower than that measured for the TT case. As a consequence the measured times for the unshuffle operation are lower for the DAC-based algorithm, as shown in the righthand plot in Fig. 3. These timings were obtained on a 12-core system composed of two Intel Xeon E5649 processors, using a gfortran compiler with the -O3 flag set. The DAC-based algorithm was parallelised using OpenMP. It is evident from the timings that the speed-up for the DAC-based algorithm is less than would be expected from Eq. 10. When the speed-ups were measured without setting the -O3 compiler flag they were much closer to the value given by Eq. 10, so it may be that compiling for OpenMP inhibits optimizations that the compiler would otherwise make. This issue is still being investigated. Note that since the shuffle and unshuffle operations carry out the same number of swaps, the results in Fig. 3 also hold for the shuffle operation.

5.2 The Use of Constant Additional Memory

The TT algorithm [4] requires additional space for m matrix elements to find A^{T} for a matrix of size $n \times m$, with $n > m$. This additional space is used in transposing the $n_i \times r_i$ matrices R_j, as mentioned in step 2 of Sect. 4.2, in performing the permutation in step 3 of Sect. 4.2, and in the shuffle and unshuffle operations. However, it appears to us that only constant additional space is required for the following reasons:

1. The $n_i \times r_i$ matrices R_j can be made as small as one pleases by the principle of infinite descent. Instead of using m as the bound for ending the recursion a constant, say 100, can be used instead.
2. The permutation vector v of step 3 of phase 2 can be avoided by instead applying the unshuffle operation $g_i - 1$ times.
3. The permutation vector v of size m used in the shuffle and unshuffle operations in Sect. 4.4 can be avoided by using a readily parallelisable divide-and-conquer approach that uses exchange operations, as explained above.

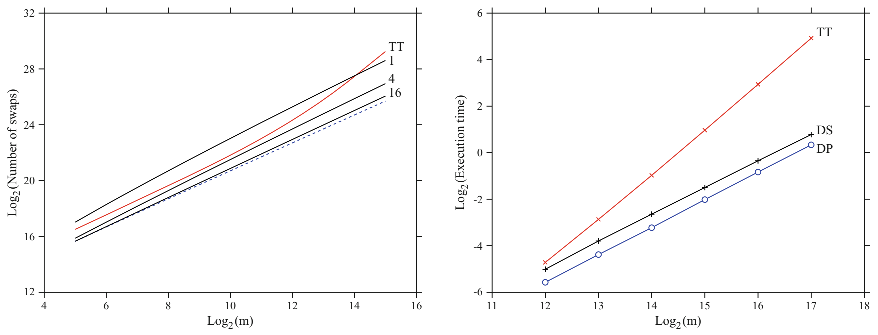


Fig. 3. Performance results as a function of problem size for the unshuffle operation for $\ell_a = 1152$ and $\ell_b = 640$. *Left*: Number of swaps. The red curve shows the results for the algorithm given by TT. The solid black lines show numbers for the DAC-based algorithm and are labelled by the number of processors, n_p. *Right*: Measured execution times. Results for the algorithm given by TT, the sequential DAC-based algorithm (DS), and the parallel DAC-based algorithm (DP) are shown.

The three items above are the only places where extra storage of length m is used. For each of them we have removed their use in our implementation.

5.3 Exploiting Parallelism

In addition to the DAC-based algorithm for the shuffle and unshuffle operations, the TT algorithm provides other opportunities for improving performance through parallelism:

1. After the matrix has been partitioned, each of the sub-matrices can then be transposed in parallel by calling routine IPSB_PowerTranspose.
2. In IPSB_PowerTranspose both loops can be parallelised. Thus, each $n_i \times m$ matrix B_j can be processed in parallel, as also can each of the g_i square $n_i \times n_i$ sub-matrices of each B_j, giving rise to $g_i m^i$-way parallelism.

5.4 Novel Algorithms for Matrix Transposition

We have investigated alternative in-place matrix transposition algorithms and compared their performance with that of the TT algorithm. The main difference between these algorithms is in how they partition the matrix. As noted in Sect. 4.1, the TT algorithm partitions an $n \times m$ matrix, with $n > m$, based on the base-m representation of m. We have developed the following alternative partitionings:

1. *GIP Transpose.* The General-In-Place Transpose partitions the matrix into q square matrices of size $m \times m$ and one other matrix of size $r \times m$, where $n = qm + r$ and $0 \leq r < m$. It is simple to transpose these square matrices in place. The remaining $r \times m$ matrix is transposed in a similar way to steps 2–4 in Fig. 2.

2. *GCD Transpose.* This variant first partitions the matrix into m matrices, B_i for $i = 0, \ldots m-1$, of size $q \times m$ and one matrix, R, of size $r \times m$, where $n = qm + r$ and $0 \leq r < m$. Each of the B_i matrices is then partitioned and transposed as in the GIP Transpose algorithm. The remaining panel matrix, R is transposed in a similar way to steps 2–4 in Fig. 2.

Both the GIP and GCD transpose algorithms can be readily parallelised as each of the matrices produced by the partitioning can be independently transposed.

We have compared the overall measured performance of the TT, GIP, and GCD in-place transpose algorithms in terms of the number of swaps and the execution time. The results are presented in Fig. 4, in which the righthand plot also shows the performance of a naive implementation of the cycle-following algorithm described in Sect. 3, where the poor performance of this approach is ascribed to its irregular memory accesses.

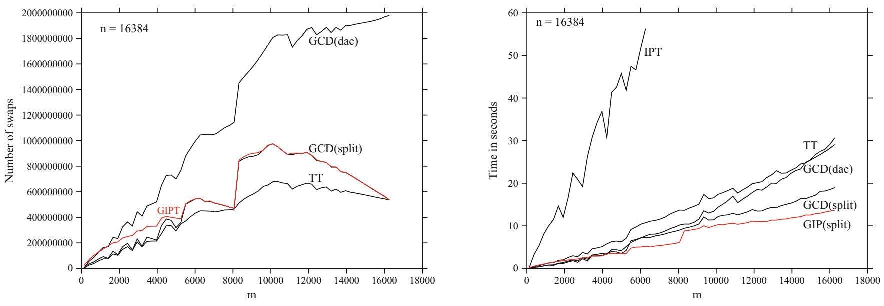


Fig. 4. Performance results for in-place transposition of an $n \times m$ matrix as a function m for $n = 16384$. *Left*: Number of swaps; *Right*: Measured execution times. The data labelled "TT" shows the results for the TT algorithm. The data labelled "GCD(dac)" is for the GCD transpose algorithm using the sequential DAC-based version of the shuffle and unshuffle operations. The data labelled "GCD(split)" is for the GCD transpose algorithm using a variant to the TT algorithm for the shuffle and unshuffle operations. The data shown in red and labelled "GIPT" is for the GIP transpose algorithm. The data labelled "IPT" in the righthand plot shows timings for a simple cycle-following algorithm.

The most interesting aspect of the plots in Fig. 4 is that although the TT algorithm has the least number of swaps it has a longer execution time than the GCD and GIP transpose algorithms for values of m greater than about 5000. We have examined the cache behavior for these algorithms using the Cachegrind tool [1], and preliminary analysis suggests that cache performance is the cause of the longer execution times seen for large m. However, further work is required to fully understand the impact of caching on the performance of these algorithms.

6 Conclusions

This paper has described in detail the swap-based algorithm of TT [4]. We have implemented this algorithm and reported on some variations that have the potential for improving performance, either by introducing parallelism, reducing the number of operations, or optimising the use of hierarchical memory. We have also described a variant of the TT algorithm that requires constant additional memory, rather than additional memory for m matrix elements. Our future work will involve understanding the parallel performance of our DAC-based implementation of the shuffle and unshuffle operations, and analysing the caching behavior of the TT, GCD and GIP transpose algorithms.

Acknowledgements. The authors are grateful to Dr. Jóse-Ramón Herrero of Universitat Politècnica de Catalunya for his helpful comments.

References

1. Cachegrind: Cachegrind description. http://valgrind.org/docs/manual/cg-manual.html (2013) Accessed 21 Nov 2013
2. Gustavson, F.G., Karlsson, L., Kagstrom, B.: Parallel and cache-efficient in-place matrix storage format conversion. ACM Trans. Math. Softw. **38**(3), 17:1–17:32 (2012)
3. Gustavson, F.G., Walker, D.W.: Proof of the swap count for the exchange operation. Technical report, School Computer Science and Informatics, Cardiff University, Cardiff, U.K (2013)
4. Tretyakov, A.A., Tyrtyshnikov, E.E.: Optimal in-place transposition of rectangular matrices. J. Complex. **25**, 377–384 (2009)

FooPar: A Functional Object Oriented Parallel Framework in Scala

Felix Palludan Hargreaves and Daniel Merkle(✉)

Department of Mathematics and Computer Science,
University of Southern Denmark, Odense, Denmark
{daniel,felhar07}@imada.sdu.dk

Abstract. We present FooPar, an extension for highly efficient Parallel Computing in the multi-paradigm programming language Scala. Scala offers concise and clean syntax and integrates functional programming features. Our framework FooPar combines these features with parallel computing techniques. FooPar is designed to be modular and supports easy access to different communication backends for distributed memory architectures as well as high performance math libraries. In this article we use it to parallelize matrix-matrix multiplication and show its scalability by a isoefficiency analysis. In addition, results based on a empirical analysis on two supercomputers are given. We achieve close-to-optimal performance wrt. theoretical peak performance. Based on this result we conclude that FooPar allows programmers to fully access Scalas design features without suffering from performance drops when compared to implementations purely based on C and MPI.

Keywords: Functional programming · Isoefficiency · Matrix multiplication

1 Introduction

Functional programming is becoming more and more ubiquitous (lambda functions introduced in C++11 and Java8) due to higher levels of abstraction, better encapsulation of mutable state, and a generally less error prone programming paradigm. In HPC settings, the usual argument against the added functional abstraction is performance issues. FooPar aims to bridge the gap between HPC and functional programming by hitting a sweet spot between abstraction and efficiency not addressed by other functional frameworks.

There exists a multitude of map-reduce based frameworks similar to Hadoop which focus on big data processing jobs, often in cloud settings. Other functional parallel frameworks like Haskell's *Eden* (semi-explicit parallel programming via *skeletons*) [13] and Scala's *Spark* [21] focus on workload balancing strategies neglecting performance to increase abstraction. While many different functional frameworks are available, most seem to value abstraction above all else. With FooPar, we reach asymptotic and practical performance goals comparable to even optimized C-code.

In this paper (after definitions and a brief introduction to isoefficiency in Sect. 2) we will introduce FooPar in Sect. 3 and describe its architecture, data structures, and operations it contains. The complexity of the individual operations on the (parallel) data structures will be shown to serve as basis for parallel complexity analysis. A matrix-matrix multiplication algorithm will be designed using the functionality of FooPar; the implementation will be analyzed with an isoefficiency analysis in Sect. 4. Test results showing that FooPar can reach close-to theoretical peak performance on large supercomputers will be presented in Sect. 5. We conclude with Sect. 6.

2 Definitions, Notations, and Isoefficiency

The most widespread model for scalability analysis of homogeneous parallel systems (i.e. the parallel algorithm and the parallel architecture) is isoefficiency [7,12,17] analysis. The *isoefficiency function* for a parallel system relates the *problem size* W and the *number of processors* p and defines how large the problem size as a function in p has to grow in order to achieve a constant pre-given efficiency. Isoefficiency has been applied to a wide range of parallel systems (see, e.g. [3,9,11]). As usual, we will define the *message passing costs*, t_c, for parallel machines as $t_c := t_s + t_w \cdot m$, where t_s is the start-up time, t_w is the per-word transfer time, and m is the message size. The *sequential (resp. parallel) runtime* will be denoted as T_S (resp. T_P). The *problem size* W is identical to the sequential runtime, i.e. $W := T_S$. The *overhead function* will be defined as $T_o(W, p) := pT_P - T_S$. The isoefficiency function for a parallel system is usually found by an algebraic reformulation of the equation $W = k \cdot T_o(W, p)$ such that W is a function in p only (see e.g. [7] for more details). In this paper we will employ broadcast and reduction operations for isoefficiency analysis for parallel matrix-matrix multiplication with FooPar. Assuming a constant cross-section bandwith of the underlying network and employing recursive doubling leads to a one-to-all broadcast computational runtime of $(t_s + t_w \cdot m) \log p$ and the identical runtime for an all-to-one reduction with any associative operation λ. All-to-all broadcast and reduction have a runtime of $t_s \log p + t_w \cdot (p - 1)$. A circular shift can be done in runtime $t_s + t_w \cdot m$ if the underlying network has a cross-section bandwith of $O(p)$.

3 The FooPar Framework

FooPar is a modular extension to Scala [16] which supports user extensions and additions to data structures with proven Scala design patterns. Scala is a *scalable language* pointing towards its ability to make user defined abstractions seem like first class citizens in the language. The object oriented aspect leads to concise and readable syntax when combined with operator overloading, e.g. in matrix operations. Scala unifies functional and imperative programming making

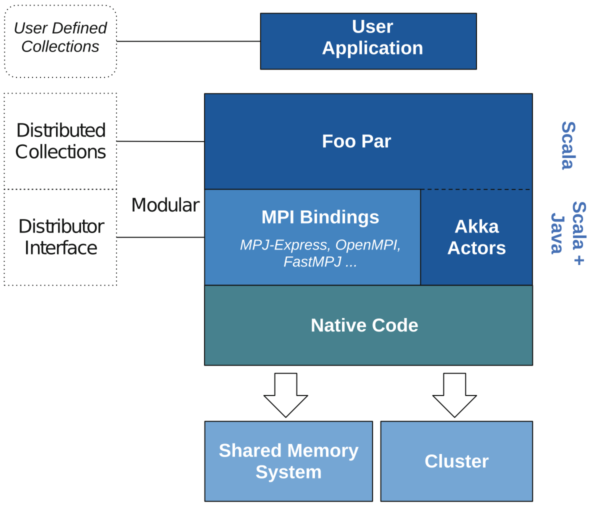


Fig. 1. Conceptional overview of the layered architecture of FooPar.

it ideal for high performance computing. It builds on the Java Virtual Machine (JVM) which is a mature platform available for all relevant architectures. Scala is completely interoperable with Java, which is one of the reasons why many companies move their performance critical code to a Scala code base [1]. Today, efficiency of byte-code can approach that of optimized C-implementations within small constants [10]. Further performance boosts can be gained by using *Java Native Interface*; however, this adds an additional linear amount of work due to memory being copied between the virtual machine and the native program. In other words, *super linear workloads* motivate the usage of *JNI*.

Figure 1 depicts the architecture of FooPar. Using the *builder/traversable* pattern [15], one can create maintainable distributed collection classes while benefiting from the underlying modular communication layer. In turn, this means that user provided data structures receive the same benefits from the remaining layers of the framework as the ones that ship with FooPar. It is possible to design a large range of parallel algorithms using purely the data structures within FooPar although one is nor restricted to that approach.

A configuration of FooPar can be described as FooPar-X-Y-Z, where X is the communication module implemented on JVM, and Y is the native code used for networking and Z is the hardware configuration, e.g. $X \in \{MPJ - Express, OpenMPI, FastMPJ, SharedMemory, Akka\}$, $Y \in \{\text{MPI, Sockets}\}$ and $Z \in \{\text{SharedMemory, Cluster, Cloud}\}$. Note that this is not an exhaustive listing of module possibilities. In this paper we only use Y = MPI and Z = Cluster and do not analyze Shared Memory parallelisation. Therefore, we will only use the notation FooPar-X.

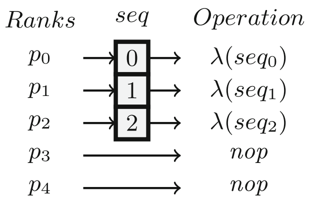


Fig. 2. A distributed map operation.

```
3:DSeq(None)
2:DSeq(Some(1))
0:DSeq(Some(0))
1:DSeq(Some(1))
4:DSeq(None)
```

Fig. 3. Output of the distributed map operation (arbitrary order).

3.1 Technologies

Currently, FooPar uses the newest version of Scala 2.10. The *Scalacheck* framework is a specification testing framework for Scala which is used to test the methods provided by FooPar data structures. *JBLAS*, a high performing linear algebra library [2] using BLAS via JNI is used to benchmark FooPar with an implementation of distributed matrix-matrix multiplication. Intel®'s *Math Kernel Library* offers an high-performing alternative with Java bindings, and will also be used for benchmarking.

3.2 SPMD Operations on Distributed Sequences

FooPar is inspired by the *SPMD/SIMD Principle* often seen in parallel hardware [4]. The *Option* monad in *Scala* is a construct similar to *Haskell's maybe monad*. Option is especially suited for SPMD patterns since it supports `map` and `foreach` operations. Listing 1.1 exemplifies this characteristic approach in FooPar. Here, `ones(i)` counts the number of 1's in the binary representation of `i`. `mapD` distributes the map operation on the Scala range `seq`.

Listing 1.1. SPMD example

```
1 def ones(i: Int): Int = i.toBinaryString.count(_ == '1') val seq = 0
2 to worldSize - 3 val counts = seq mapD ones
3 println(globalRank+":"+counts)
```

In SPMD, every process runs the same program, i.e. every process generates `seq` in line 3. If combined with lazy-data objects, this does not lead to unnecessary space or complexity overhead (cmp. Figs. 2 and 3). While every process generates the sequence, only *some* processes perform the `mapD` operation.

3.3 Data Structures

FooPar relies heavily on the interpretation of data structures as *process-data* mappings. As opposed to many modern parallel programming tools, FooPar uses static mappings defined by the data structures and relies on the user to

Table 1. A selection of operations on distributed sequences in FooPar.

Operation	Semantic	Notes	T_p (parallel runtime)
`mapD`(λ)	*Each process* transforms one element of the sequence using operation λ (element size m)	This is a non-communicating operation	$\Theta(T_\lambda(m))$
`reduceD`(λ)	The sequence with p elements is reduced to the *root process* using operation λ	λ must be an *associative* operator	$\Theta(\log p(t_s + t_w m + T_\lambda(m)))$
`allGatherD`	All processes obtain a list where element i comes from process i	Process i provides *the valid* ith element	$\Theta((t_s + t_w m)(p-1))$
`apply(i)`	All processes obtain the ith element of the sequence	sementically identical to a one-to-all broadcats	$\Theta(\log p(t_s + t_w m))$

partition input. This decision was made to ensure efficiency and analyzability. By using static mappings in conjunction with *SPMD*, the overhead and bottleneck pitfalls induced by *master worker* models are avoided and program-simplicity and efficiency are achieved. In FooPar, data partitioning is achieved through *proxy-* or *lazy* objects, which are easily defined in *Scala*. In its current state, FooPar supports distributed singletons (aka. distributed variables), distributed sequences and distributed multidimensional sequences. The distributed sequence combines the notion of *communication groups* and data. By allowing the dynamic creation of communication groups for sequences, a total abstraction of network communication is achieved. Furthermore, a communication group follows data structures for subsequent operations allowing for advanced chained functional programming to be highly parallelized. Table 1 lists a selection of supported operations on distributed sequences. The given runtimes are actually achieved in FooPar, but of course they depend on the implementation of collective operations in the communication backend. A great advantage of excluding user defined message passing is gaining analyzability through the provided data-structures.

4 Matrix-Matrix Multiplication in FooPar

4.1 Serial Matrix-Matrix Multiplication

Due to the abstraction level provided by the framework, algorithms can be defined in a fashion which is often very similar to a mathematical definition. Matrix-matrix multiplication is a good example of this. The problem can be

```
//Initialize matrices
val A = Array.fill(M, M)(MJBLProxy(SEED, b))
val Bt = Array.fill(M, M)(MJBLProxy(SEED, b)).transpose

//Multiply matrices
for (i <- 0 until M; j <- 0 until N)
  A(i) zip Bt(j) mapD { case (a, b) => a * b } reduceD (_ + _)
```

Algorithm 1: Generic algorithm for matrix-matrix multiplication with FooPar.

defined as $(AB)_{i,j} := \sum_{k=0}^{n-1} A_{i,k}B_{k,j}$, where n is the number of rows and columns in matrices A and B respectively. In functional programming, list-operations can be used to model this expression in a concise manner. The three methods, *zip, map* and *reduce* are enough to express matrix-matrix multiplication as a functional program (for an introduction to functional programming concepts refer to [20]). A serial algorithm for matrix-matrix multiplication based on a 2d-decomposition of the matrices could look like this:

$$C_{i,j} \leftarrow \mathit{reduce}\ (+)\ (\mathit{zipWith}\ (\cdot)\ A_{i*}\ B^T_{j*}), \qquad \forall(i,j) \in \mathcal{R} \times \mathcal{R} \qquad (1)$$

Here, $\mathcal{R} = \{0, \ldots, q-1\}$, and the sub-matrices are of size $(n/q)^2$. Operation *zipWith* is a convenience method roughly equivalent to: $\mathit{map} \circ \mathit{zip}$, which takes 2 lists and a 2-arity function to combine them.

4.2 Generic Algorithm for Parallel Matrix-Matrix Multiplication

To illustrate the simplicity of complexity analysis, the parallel version of the algorithm can be written in a more verbose fashion as follows:

$$C_{i,j} \leftarrow \texttt{reduceD}\ (+)\ (\texttt{mapD}\ (\cdot)\ (\texttt{zip}\ A_{i*}\ B^T_{*j})), \qquad \forall(i,j) \in \mathcal{R} \times \mathcal{R} \qquad (2)$$

Operation `zip` is $\in \Theta(1)$ due to lazy evaluation. We use a block size $m = (n/q)^2$. For `mapD` (multiplication of sub-matrices) we have $T_{\text{mult}}(m) = \Theta(m^{3/2})$, for `reduceD` (summation of sub-matrices) we have $T_{\text{sum}}(m) = \Theta(m)$. In asymptotic terms the *parallel runtime* T_P is therefore:

$$T_P = \overbrace{\Theta(1)}^{\texttt{zip}} + \overbrace{\Theta((n/q)^3)}^{\texttt{mapD}} + \overbrace{\Theta((n/q)^2 \log q)}^{\texttt{reduceD}}$$

Since $C_{i,j}$ is independent both in i and j, the q^2 operations can all run in parallel. Using q processors per reduction leads then to $p = q^2 \cdot q$ processors and the overall asymptotic runtime $\Theta((n/p)^2 \log p)$.

Using the framework, some parts of the analysis can be carried out independently of the *lambda operations* used in an algorithm. What is left is a *generic algorithm* which shows precisely the communication pattern used in the algorithm. As a coincidence, the communication pattern is essentially identical to that of the well known DNS algorithm [5,9].

Algorithm 1 shows a complete FooPar implementation, which is practically identical to the pseudo code. Note, that the algorithm uses proxy-objects which are simply objects containing lazy data using Scala's *lazy* construct [14].

Isoefficiency Analysis for the Generic Algorithm: We start by determining the non-asymptotic parallel runtime. We assume the number of processors is $p = q^3$ (i.e. $q = p^{1/3}$) and matrices A and B of size $n \times n$. Splitting A and B into $q \times q$ blocks leads to a block size of $(n/q)^2$. The `zip` operation has a runtime of q^2 due to `nop` instructions carried out in iterations where the current process is not assigned to the operation. An implicit conversion (runtime q^2) is needed to extend the functionality of standard Scala arrays. The `mapD` operation has a runtime of $q^2 + (n/q)^3$ and the `reduceD` operation has a runtime of $q^2 + \log q + (n/q)^2 \log q$. As $q^2 = p^{2/3}$, this leads to an overall parallel runtime of

$$T_p = 4 \cdot p^{2/3} + \frac{n^3}{p} + 1/3 \left(\log p + \left(\frac{n^2}{p^{2/3}} \right) \log p \right),$$

and the corresponding cost $p \cdot T_P \in \Theta(4p^{5/3} + n^3)$. Therefore this approach is cost-optimal for $p \in O(n^{9/5})$. The overhead for this basic implementation is

$$T_o = pT_p - T_S = 4p^{5/3} + \frac{p}{3} \left(\log p + \left(\frac{n^2}{p^{2/3}} \right) \log p \right).$$

Following an isoefficiency analysis based on $W = K \cdot T_o(W, p)$ leads to

$$W = n^3 = K4p^{5/3} + Kp \left(\log p + \left(\frac{n^2}{p^{2/3}} \right) \log p \right).$$

Examining the terms individually shows that the first term of $K \cdot T_o(W, p)$ constraints the scalability the most. Therefore, the isoefficiency function for the basic algorithm is $W \in \Theta(p^{5/3})$. Figure 4 shows the communication pattern implemented by Algorithm 1.

4.3 Grid Abstraction in FooPar for Parallel Matrix-Matrix Multiplication

In [8] an isoefficiency function in the order of $\Theta(p \log^3 p)$ was achieved by using the DNS algorithm for matrix-matrix multiplication. The bottleneck encountered in the basic implementation is due to the inherently sequential `for loop` emulating the $\forall$ quantifier. Though Scala offers a lot of support for *library-as-DSL* like patterns, there is no clear way to offer safe parallelisation of nested for loops while still supporting distributed operations on data structures. To combat this problem, FooPar supports multidimensional distributed sequences in conjunction with constructors for arbitrary Cartesian grids. `Grid3D` is a special case of `GridN`, which supports iterating over 3D-tuples as opposed to *coordinate lists*. Using `Grid3D` an algorithm for matrix-matrix multiplication can be implemented

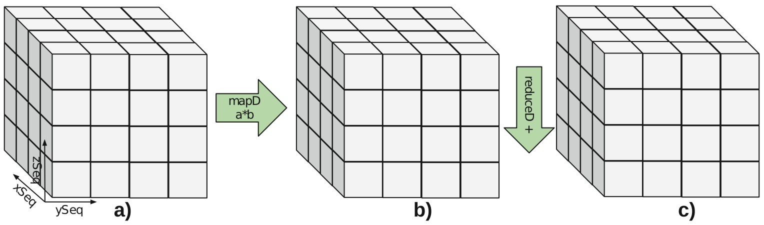


Fig. 4. **(a)** Process (i, j, k) contains blocks $A_{i,k}$ and $B_{k,j}$ **(b)** local multiplication $C_{i,j} = A_{i,k} \times B_{k,j}$, **(c)** reduction (summation): $(i, j, 0)$ contains the (partial) result matrix.

```
val R = 0 until DIM
val G = Grid3D(R, R, R)

val GA = G mapD { case (i, j, k) => A(i)(k) }
val GB = G mapD { case (i, j, k) => B(k)(j) }

val C = ((GA zipWithD GB)(_ * _) zSeq) reduceD (_ + _)
```

Algorithm 2: Matrix-matrix multiplication in FooPar using Grid Abstraction.

as seen in Algorithm 2. `zSeq` is a convenience method for getting the distributed sequence, which is variable in z and constant in the x, y coordinates of the current process. By using the grid data structure, we safely eliminate the overhead induced by the for-loop in Algorithm 1 and end up with the same basic communication pattern as shown in Fig. 4. Operation `mapD` has a runtime of $\Theta((n/q)^3)$ and `reduceD` a runtime of $\Theta(\log q + (n/q)^2 \log p)$. Due to space limitations we will not present the details of runtime and isoefficiency analysis but refer to [9], as the analysis given there is very similar. Parallel runtime, T_P, and cost are given by $T_P = n^3/p + \log p + \left(n^2/p^{2/3}\right) \log p$ and cost $\in \Theta(n^3 + p \log p + n^2 p^{1/3} \log p)$. This leads to an isoefficiency function in the order of $\Theta(p \log^3 p)$, identical to the isoefficiency achieved by the DNS algorithm.

5 Test Results

Parallel Systems and their Interconnection Framework: In this study we focus on analyzing scalability, efficiency and flexibility. We tested FooPar on two parallel systems: the first system is called Carver and is used to analyze the *peak performance* and the overhead of FooPar. It is an IBM iDataPlex system where each computing node consists of two Intel Nehalem quad-core processors (2.67 GHz processors, each node has at least 24 GB of RAM). The system is located at the Department of Energy's National Energy Research Scientific

Computing Center (NERSC). All nodes are interconnected by 4X QDR InfiniBand technology, providing maximally 32 Gb/s of point-to-point bandwidth. A highly optimized version of Intel's Math Kernel Library (MKL) is used, which provides an empirical peak performance of 10.11 GFlop/s on one core (based on a single core matrix-matrix multiplication in C using MKL). This will be our reference performance to determine efficiency on Carver. Note, that the empirical peak performance is very close to the theoretical peak performance of 10.67 GFlop/s on one node. The largest parallel job in Carver's queuing system can use maximally 512 cores, i.e. the theoretical peak is 5.46 TFlop/s.

The second system has basically the same hardware setup. The name of the system is Horseshoe-6 and it is located at the University of Southern Denmark. Horseshoe-6 is used in order to test the *flexibility* of FooPar. The math libraries are not compiled towards the node's architecture, but a standard high performing BLAS library was employed for linear algebraic operations. The reference performance on one core was measured again by a matrix-matrix multiplication (C-version using BLAS) and is 4.55 GFlop/s per core.

On Carver Java bindings of the nightly-build OpenMPI version 1.9a1r27897 [6] were used in order to interface to OpenMPI (these Java bindings are not yet available in the stable version of OpenMPI). On Horseshoe-6 we used three different communication backends, namely (i) OpenMPI Java bindings (same version as on Carver), (ii) MPJ-Express [18], and (iii) FastMPJ [19]. Note, that changing the communication backend does not require any change in the Scala source code for the parallel algorithmic development within FooPar.

For performance comparison of FooPar and C we also developed a highly optimized parallel version of the DNS algorithm for matrix-matrix multiplication, using C/MPI. MKL (resp. BLAS) was used on Carver (resp. Horseshoe-6) for the sub-matrix-matrix multiplication on the individual cores. Note, that the given efficiency results basically do not suffer any noticable fluctuations when repeated.

Results on Carver: Efficiencies for different matrix sizes, n, and number of cores, p, are given in Fig. 5. As communication backend, we used OpenMPI. We note that we improved the Java implementation of `MPI_Reduce` in OpenMPI: the nightly build version implements an unnecessarily simplistic reduction with $\Theta(p)$ send/receive calls, although this can be realized with $\Theta(\log p)$ calls. I.e., the unmodified OpenMPI does *not* interface to the native `MPI_Reduce` function, and therefore introduces an unnecessary bottleneck.

For matrix sizes $n = 40000$ and the largest number of cores possible (i.e. $p = 512$) Algorithm 2 achieves 4.84 TFlop/s, corresponding to 88.8 % efficiency w.r.t. the theoretical peak performance (i.e. 93.7 % of the empirically achievable peak performance) of Carver. The C-version performs only slightly better. Note, that the stronger efficiency drop (when compared to Horseshoe-6 results for smaller matrices) is due to the high performing math libraries; the absolute performance is still better by a factor of ≈2.2. We conclude that the computation

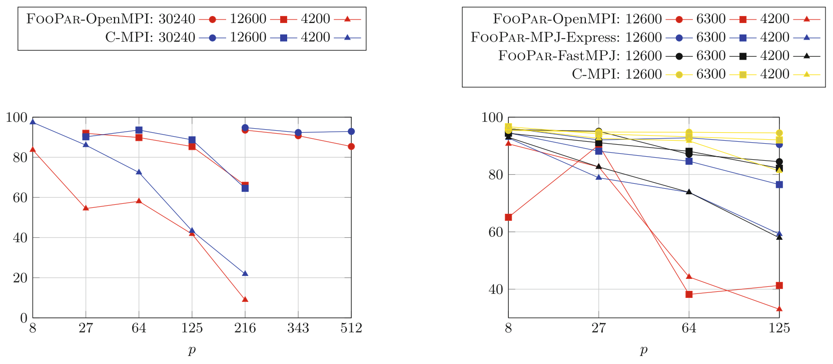


Fig. 5. Efficiency results for matrix-matrix multiplication (size $n \times n$) with Grid Abstraction; x-axis: number of cores used; the value for n and the communication backend employed are given in the legend. Left: results on Carver, Right: results on Horseshoe-6; efficiency is given relative to empirical peak performance on one core (see text).

and communication overhead of using FooPar is neglectable for practical purposes. While keeping the advantages of higher-level constructs, we manage to keep the efficiency very high. This result is in line with the isoefficiency analysis of FooPar in Sect. 4.

Results on Horseshoe-6: On Horseshoe-6 we observed that the different backends lead to rather different efficiencies. When using the unmodified OpenMPI as a communication backend, a performance drop is seen, as expected, due to the reasons mentioned above. Also MPJ-Express uses an unnecessary $\Theta(p)$ reduction (FastMPJ is closed source). However, if FooPar will not be used in an HPC setting and efficiency is not be the main objective (like in a heterogeneous system or a cloud environment), the advantages of "slower" backends (like running in daemon mode) might pay off.

6 Conclusions

We introduced FooPar, a functional and object-oriented framework that combines two orthogonal scalabilities, namely the scalability as seen from the perspective of the Scala programming language and the scalability as seen from the HPC perspective. FooPar allows for isoefficiency analyses of algorithms such that theoretical scalability behavior can be shown. We presented parallel solutions in FooPar for matrix-matrix multiplication and supported the theoretical finding with empirical tests that reached close-to-optimal performance w.r.t. the theoretical peak performance on 512 cores.

Acknowledgments. We acknowledge the support of the Danish Council for Independent Research, the Innovation Center Denmark, the Lawrence Berkeley National Laboratory, and the Scientific Discovery through Advanced Computing (SciDAC) Outreach Center. We thank Jakob L. Andersen for supplying a template C-implementation of the DNS algorithm.

References

1. Scala in the enterprise. Ecole Polytechnique Federale de Lausanne (EPFL). http://www.scala-lang.org/node/1658 (2013). Accessed 4 May 2013
2. Abeles, P.: Java-Matrix-Benchmark - a benchmark for computational efficiency, memory usage and stability of Java matrix libraries. http://code.google.com/p/java-matrix-benchmark/ (2013). Accessed 12 Feb 2013
3. Bosque, J.L., Robles, O.D., Toharia, P., Pastor, L.: H-Isoefficiency: scalability metric for heterogenous systems. In: Proceedings of the 10th International Conference of Computational and Mathematical Methods in Science and Engineering (CEMMSE 2010), pp. 240–250 (2010)
4. Darema, F.: The SPMD model: past, present and future. In: Cotronis, Y., Dongarra, J. (eds.) EuroPVM/MPI 2001. LNCS, vol. 2131, p. 1. Springer, Heidelberg (2001)
5. Dekel, E., Nassimi, D., Sahni, S.: Parallel matrix and graph algorithms. SIAM J. Comput. **10**(4), 657–675 (1981)
6. Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L., Woodall, T.S.: Open MPI: Goals, concept, and design of a next generation MPI implementation. In: Proceedings of the 11th European PVM/MPI Users' Group Meeting, pp. 97–104 (2004)
7. Grama, A., Gupta, A., Kumar, V.: Isoefficiency: measuring the scalability of parallel algorithms and architectures. IEEE Parallel Distrib. Technol. Syst. Appl. **1**(3), 12–21 (1993)
8. Grama, A., Karypis, G., Kumar, V., Gupta, A.: Introduction to Parallel Computing. Addison Wesley, Reading (2003)
9. Gupta, A., Kumar, V.: Scalability of parallel algorithms for matrix multiplication. In: Proceedings of the 22nd International Conference on Parallel Processing, ICPP, vol. 3, pp. 115–123 (1993)
10. Hundt, R.: Loop recognition in C++/Java/Go/Scala. In: Proceedings of Scala Days (2011)
11. Hwang, K., Xu, Z.: Scalable Parallel Computing. McGraw-Hill, New York (1998)
12. Kumar, V., Rao, V.N.: Parallel depth first search, part II: analysis. Int. J. Parallel Prog. **16**(6), 501–519 (1987)
13. Loogen, R., Ortega-Mallén, Y., Peña, R.: Parallel functional programming in Eden. J. Funct. Program. **15**, 431–475 (2005)
14. Odersky, M.: The Scala language specification (2011)
15. Odersky, M., Moors, A.: Fighting bit rot with types (experience report: Scala collections). In: Proceedings of the 29th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009), vol. 4 of Leibniz International Proceedings in Informatics, pp. 427–451 (2009)
16. Odersky, M., Spoon, L., Venners, B.: Programming in Scala. Artima, New York (2010)

17. Quinn, M.J.: Parallel Programming in C with MPI and OpenMP. McGraw-Hill, Blacklick (2003)
18. Shafi, A., Manzoor, J.: Towards efficient shared memory communications in MPJ express. In: Proceedings of the 25th IEEE International Symposium on Parallel Distributed Processing 2009 (IPDPS), pp. 1–7 (2009)
19. Taboada, G.L., Touriño, J., Doallo, R.: F-MPJ: scalable Java message-passing communications on parallel systems. J. Supercomput. **1**, 117–140 (2012)
20. Thompson, S.J., Wadler, P.: Functional programming in education - introduction. J. Funct. Program. **3**(1), 3–4 (1993)
21. Zaharia, M., Chowdhury, N.M.M., Franklin, M., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. Technical Report UCB/EECS-2010-53, EECS Department, University of California, Berkeley (2010)

Effects of Segmented Finite Difference Time Domain on GPU

Jose Juan Mijares Chan[1,2], Gagan Battoo[1], Parimala Thulasiraman[1](✉), and Ruppa K. Thulasiram[1]

[1] Department of Computer Science, University of Manitoba, Winnipeg, MB, Canada
[2] Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada
thulasir@cs.umanitoba.ca

Abstract. Finite Difference Time Domain (FDTD) is the most popular method in computational electromagnetics. In acoustics, FDTD is often used as a numerical analysis technique to model mechanical wave and acoustics. FDTD in general is computationally expensive in terms of time due to its large number of time steps for accurate precision and is data parallel in nature. However, it is also memory bounded. Although previous work on FDTD has studied the effect of parallelizing FDTD on accelerators to reduce computational cost, the memory bounded problem has not been studied. In this work we consider the segmented FDTD (SFDTD) algorithm that divides the problem space into segments to reduce computational redundancy and also reduce memory. We exploit the memory hierarchy of the GPU to efficiently implement the SFDTD algorithm. To the best of our knowledge, this is the first work that studies the implementation of the SFDTD algorithm on GPU and its effect on memory.

Keywords: GPU · CUDA · Segmented FDTD · Acoustics

1 Introduction

Among the numerical analysis techniques, Finite Difference Time Domain (FDTD) is the most popular method in computational electromagnetic. In 1966, Yee [25] presented the basis of FDTD numerical technique for solving Maxwell's curl equation. In 1977, Holland [7] applied Yee's algorithm to electromagnetic pulse coupling problems. Taflove [20] in 1980 coined the descriptor Finite Difference Time Domain (FDTD). Since then, a large number of publications in applying FDTD has cover many different problems such as in microwave circuits, optics, antenna and radiation problems. FDTD finds its applications in the field of "Acoustic Engineering" (a branch that deals with sound and vibration) as well. In this paper, we focus on applying FDTD to acoustic wave propagation.

In acoustic wave propagation, FDTD is used as a numerical analysis technique to model a mechanical wave and its acoustics. This phenomenon is described by two first order acoustic differential equations. One of the equations is a variation of Newton's second law. This equation considers small signals that is described in terms of pressure and velocity where pressure is a scalar field and velocity is a vector field. In acoustics, there is one vector field and one scalar field unlike two vector fields in electromagnetics. A detailed explanation of FDTD in acoustics is given in Sect. 2.

One of the important considerations in FDTD simulation is the proper truncation of finite simulation domain to approximate infinite space. Truncating boundaries may cause reflections. Therefore, proper boundary conditions are needed to minimize these reflections from the truncated boundary. Several absorbing boundary conditions [1,5,13,16,17] were developed for this purpose. One of the well-known absorbing boundary condition for electromagnetic waves is the perfectly matched layer (PML) by Berenger [1]. The reduction in reflection depends on parameters such as the grid spacing, the attenuation factor, the length of the PML zone,time-step and precision. Finer grid spacing, for example, reduces reflections to a large extent with a high computational cost. With the computational resources of multicore architectures, we can take advantage of finer grid spacing. In this work, we use PML for acoustic [27].

In general, FDTD is computationally intense in terms of time due to its large number of time steps for accurate precision and is also memory bound due to large data size [23]. There has been several works demonstrating the use of parallel processing in reducing the computational cost. Ju [9] and Yu [26] parallelize FDTD on a Beowulf cluster of 16 PCs using MPI.

FDTD can be categorized as a regular algorithm. The data structure used is usually a multi-dimensional array. At each time step, the data elements within an array can be calculated independently exhibiting large amount of parallelism, suitable for data parallel SIMT architectures. Recently, Xu et al. [24] used FDTD as a kernel in the microwave tomography algorithm. They implemented the algorithm on Cell BE. The electromagnetic community has studied the implementation of FDTD and its variations on Nvidia GPU [3,8,15,18,21,28] showing a reduction on the computational cost or execution time of the FDTD simulation by using GPUs. However, on the GPU, the number of data transfers and long memory latency impact drastically the algorithmic performance.

There has been some work on developing memory-efficient FDTD algorithms [11]. In 2008, Wu et al. [22] proposed an algorithm to reduce 3D FDTD problem to 2D and showed that this algorithm reduces computational requirements for large scale problems. Following this work, the authors proposed an improvement to their original algorithm by dividing the problem space into segments to further reduce computational redundancy and also reduce memory [23]. They called their algorithm, Segmented FDTD (SFDTD). They show that the CPU execution time increases linearly and not exponentially as dictated by the standard FDTD algorithm.

In this paper, we study SFDTD simulation with PML as the absorbing boundary condition on a Nvidia GPU, GeForce GTX 260, for acoustic wave propagation. We use the various memory features of the GPU and discuss the effects of varying the segment size and its impact on memory usage. To our knowledge this is the first work that studies the implementation of the SFDTD algorithm on GPU. This work tests the feasibility of SFDTD on GPU. Therefore, as our initial study we consider 1D SFDTD.

2 The Finite Difference Time Domain Method

The FDTD method is a precise modelling technique for acoustic simulations. The principal benefit of FDTD is the possibility of modelling the wave propagation over dense meshes with multiple materials. But in some cases, the method demands intensive calculations and the full knowledge of the properties of all the objects interacting in the simulation [12, Chapter 3].

The wave propagation phenomena involves particles compression or relaxation denoting changes on pressure and velocity as a function of time and space. For simplicity, we consider the special case of homogeneous isotropic mediums with no losses, where the velocity of the wave is kept almost constant. The wave propagation is explained by two laws, the conservation of mass and the conservation of momentum [19, Chapter 6] which are expressed as,

$$\frac{\partial \mathbf{v}}{\partial t} = -\frac{1}{\rho}\nabla p \qquad\qquad \frac{\partial p}{\partial t} = -\kappa \nabla \mathbf{v} \tag{1}$$

where p denotes the acoustic pressure, $\mathbf{v}$ the vector velocity, ρ is the density of the medium, while $\kappa = \rho c^2$ is the bulk modulus. As mentioned in the previous section, we use PML to reduce the effect of reflections at the boundaries [1]. Using Yee's staggered mesh and central difference approximation [4] together with PML, the pair of Eqs. (1) are modified to,

$$v_{i+\frac{1}{2}}^{n+\frac{1}{2}} = \begin{cases} v_{i+\frac{1}{2}}^{n-\frac{1}{2}} - \dfrac{\Delta t}{\rho \Delta x}\left(p_{i+1}^{n} - p_i^n\right) & i \in (L, N-L), \quad \text{(2a)} \\ v_{i+\frac{1}{2}}^{n-\frac{1}{2}} e^{-\kappa\alpha\Delta t} - \dfrac{1 - e^{-\kappa\alpha\Delta t}}{\kappa\alpha\rho\Delta x}\left(p_{i+1}^{n} - p_i^n\right) & \text{else.} \quad \text{(2b)} \end{cases}$$

$$p_i^{n+1} = \begin{cases} p_i^n - \kappa\dfrac{\Delta t}{\Delta x}\left(v_{i+\frac{1}{2}}^{n+\frac{1}{2}} - v_{i-\frac{1}{2}}^{n+\frac{1}{2}}\right) & i \in (L, N-L), \quad \text{(3a)} \\ p_i^n e^{-\kappa\alpha\Delta t} - \dfrac{1 - e^{-\kappa\alpha\Delta t}}{\alpha\Delta x}\left(v_{i+\frac{1}{2}}^{n+\frac{1}{2}} - v_{i-\frac{1}{2}}^{n+\frac{1}{2}}\right) & \text{else.} \quad \text{(3b)} \end{cases}$$

here, the simulation is performed over a space of size $N\Delta x$ and within a maximum time of T seconds. The simulation space is represented with N points uniformly distributed by Δx, where the spatial position i is located at $x_i = i\Delta x$ from the origin. Similarly for the simulation time, a uniformly distributed grid is employed using a Δt spacing, where the temporal index n corresponds to the point in time $t_n = n\Delta t$. The PML regions are located at the boundaries of the simulation space. With a length of $L\Delta x$, the PML region can have a size of $0 < L\Delta x < \frac{N\Delta x}{2}$, and behaves according to Eqs. (2b,3b) with an attenuation factor of α.

The simulation stability is achieved by fulfilling the Courant-Friedrich-Lewy (CFL) Criterion [2]. This rule of thumb consists of defining the upper limit for the time step in terms of the velocity of sound in the medium, v_m, and the grid spacing in 1D. This is $\Delta t = \min\left\{\frac{\Delta x}{v_m}\right\}$.

3 Segmented FDTD and Its Implementation on GPU

In algorithms like FDTD using staggered grids, the memory is limited to the simulation capacity and the space availability for storing results; also, when FDTD is applied to pulsed waves, the information concentrates in spots instead of spreading through the simulation space. Recently, among the electromagnetics community, the SFDTD was proposed for simulating acoustics in tubular shapes probing to reduce the usage of memory during simulation [23].

The main idea of SFDTD is to divide the simulation space and evaluate the FDTD of a segment where the pulsed wave is present, and then pass the necessary information to continue evaluating it in the next segment as the wave travels to the next segment. This is achieved by a establishing a set of segments, S, where the k-th segment, S_k, is a section of space where the wave will propagate from the interface C_k^{start} to C_k^{end}, passing through a block, B_k, as shown in Fig. 1. In order to sustain continuity, the minimum size of the interfaces has to be larger than a cycle from the lowest wave frequency expected in the simulation.

To transfer the wave captured on S_k to another segment, a steady state must be achieved as the wave is at C_k^{end}. This is inspected by evaluating an error function [14, Chapter 2] of S_k excluding C_k^{end}. Letting $D_{CB} = C_k^{start} \cup B_k$ and applying a difference operator using central difference method, $D = |\Delta D_{CB}|$, then the following summation becomes the error function,

$$\sum D \leq Th, \tag{4}$$

where Th is an educated guess used to compare if the pulsed wave is located at C_k^{end}. If Eq. (4) is true, the wave in C_k^{end} is copied into C_{k+1}^{start}, then S_k is erased allowing to reuse the memory space of S_k. This type of problem suits GPUs architectures due to its simple operations and data parallelism.

In our implementation, two issues were addressed, the flow dependency and memory access latency. The flow dependency can be observed in the calculation of the velocity (2a) and pressure (3a) equations. It causes race conditions among threads reading and writing on the velocity points. A solution is to solve it through synchronization, but it is at expenses of depreciating the performance. To avoid the memory access latency a faster memory banks, such as registers, can be used instead. Solving both issues, the calculations of each dependent velocity

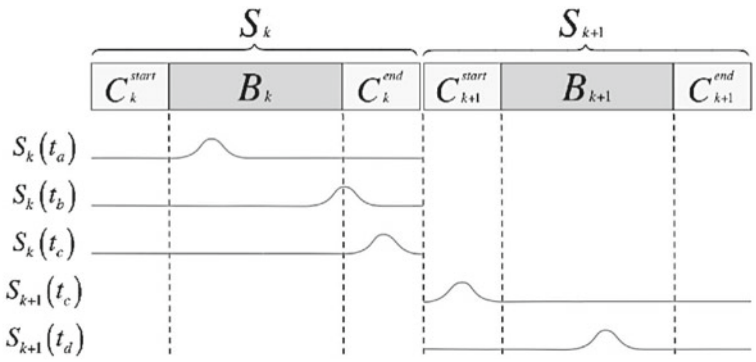


Fig. 1. Segmented simulation space and simulation states

points is stored in registers, as in wait-free algorithms [6], thereby avoiding race condition and reducing memory access latency as much as possible.

In Algorithm 1, we present the implementation of SFDTD using two segments, dS_0 and dS_1. From line 2–10, the initialization of the memory spaces for the two segments is done in the GPU and CPU. In line 10 to 19, it encompasses a simulation within the time index n, and upon completion this counter is been incremented on line 17. In line 12, the FDTD with its PML implementation gets calculated based on Eqs. (2a,2b,3a,3b); once done, the space formed by $dC_0^{start} \cup dB_0$ gets tested for the steady state as defined in Eq. (4). If it is false, dS_0 will be transfer to the CPU; otherwise, the wave at dC_0^{end} is transferred to dC_1^{start} in dS_1, and then dS_1 is transferred to dS_0, so the memory space is reused. At the same time dS_1 is transferred to the CPU to keep track of it. Then, in line 18, the simulation state for iteration n is written to the hard drive.

4 Results

In order to compare the performance of the SFDTD and the FDTD, a group of experiments were conducted on a GeForce GTX 260 connected through a PCIe v2 x16 bus. The experiments encompassed the FDTD and the SFDTD evaluation for an iteration, which corresponds to lines 12–18 at Algorithm 1. The experiments focused on different segment sizes, reviewing (1) the kernel execution time, (2) the influences on the saving to disk time, (3) the communication transfer time between CPU and GPU, (4) and a review of the total execution time and speed-up factor of the algorithms. The simulation length was selected based on the maximum off-chip memory available on the GPU, in this case approximately 895 MB. This allows us to execute upto 55M-points using IEEE 754 floating point representation, where each pair of points represent the velocity (2a,2b) and pressure (3a,3b) points previously discussed. In the case of the SFDTD, the segment size was defined to be 32, 1k, 32k, and 1M, in an attempt to investigate the effect of changing the segment size from small to large.

Algorithm 1. GPU implementation of Segmented FDTD

```
Begin
    Let $n \leftarrow 0$ be the time index
    Let $\Delta t$ be the time-step in the simulation
    Let $T$ be the Maximum time of simulation
    Let $S_0$ and $S_1$ be segments in the CPU
    Let $dS_0$ and $dS_1$ be segments in the GPU
    Let $dB_0$ be a block from $dS_0$ in the GPU
    Let $dC_0^{start}$ and $dC_0^{end}$ be the interfaces from $dS_0$ in the GPU
    Let $dC_1^{start}$ and $dC_1^{end}$ be the interfaces from $dS_1$ in the GPU
    Memory transfer from $S_0$ to $dS_0$
    While $n\Delta t < T$
        Calculate $FDTD(\ dS_0)$
        If $Steady\ State\left(\ dC_0^{start} \cup \ dB_0\right) = \ false$
            Memory transfer $dS_0$ to $S_0$
        Else
            Memory transfer $dC_0^{end}$ to $dC_1^{start}$, $dS_1$ to $S_0$, $dS_1$ to $dS_0$
        Let $n \leftarrow \ n+1$
        $Save(\ S_0)$
End
```

The way that CUDA architecture allows fine-grained data and thread parallelism nested within a coarse-grained data and task parallelism [10], allows both algorithms to be used efficiently. For both, the number of threads selected per block was 32, and the number of blocks were adjusted so each threads calculates a pressure and velocity point. For both methods, the results were averaged over 30 iterations.

Kernel Execution Time. As shown in Fig. 2, the FDTD processing time grows linearly at rate of 0.706 ms/point; meanwhile, the SFDTD remains almost constant, independently of the segment size. This is because the FDTD algorithm covers the complete simulation length within an iteration, but the SFDTD just covers the segment where the wave is spotted.

Writing to Hard Drive Time. The results in Fig. 3 show how the compacted structure of the SFDTD allows almost constant operation time, meanwhile the FDTD grows as the simulation length grows.

Communication Transfer Time. For both methods, the communication transfer time between CPU and GPU is limited to the lowest speed element on the communication path, the PCIe bus with a bandwidth of 3.0 GBs^{-1}. From Fig. 4, the SFDTD shows to perform better combining large simulations lengths and small segment sizes.

Total Execution Time. Considering that in Algorithm 1 and counting the delay for synchronization as part of each instruction, the iteration execution time can be described as, $T_{exec} = T_{kernel} + 2T_{comm} + T_{storing}$ where T_{kernel} is the time

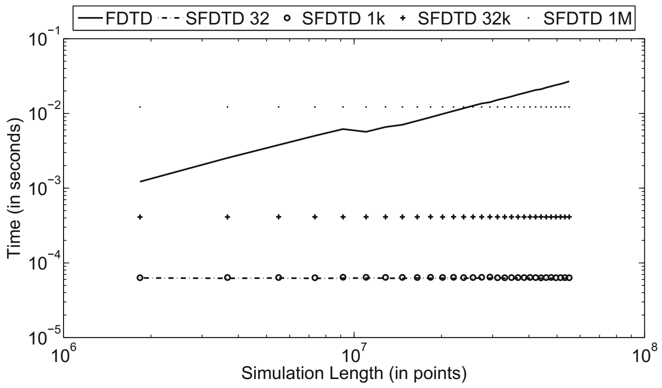


Fig. 2. Average kernel execution time for FDTD and SFDTD

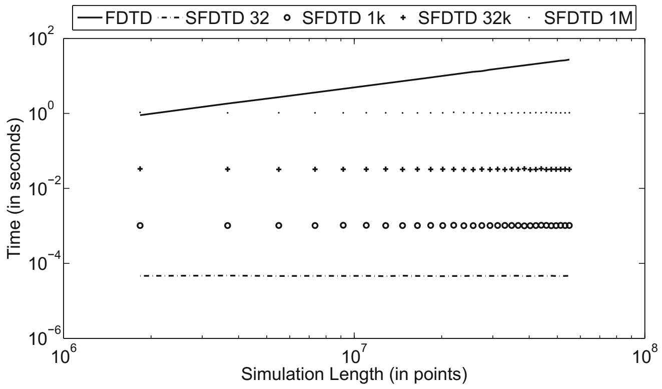


Fig. 3. Average writing to hard drive time for FDTD and SFDTD

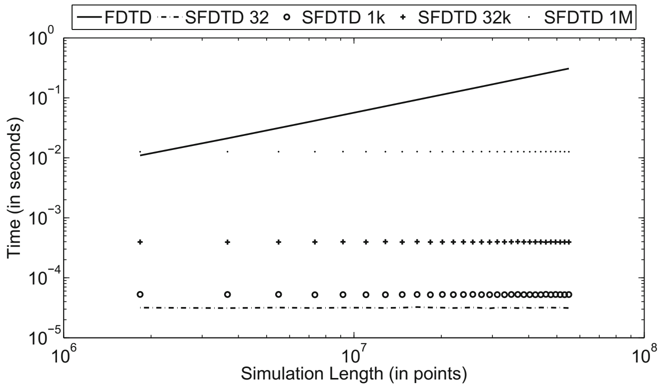


Fig. 4. Average communication transfer time from CPU to GPU and vice versa

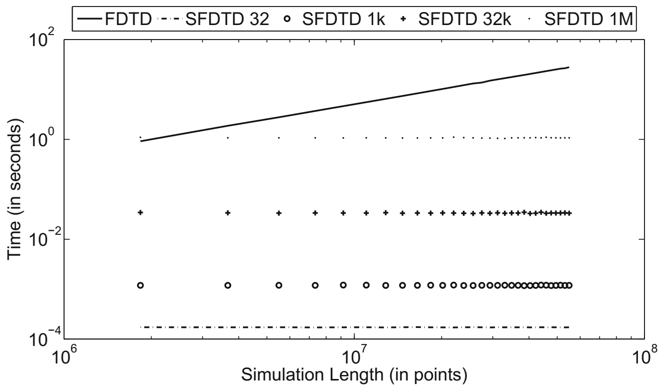


Fig. 5. Average total execution time for FDTD and SFDTD

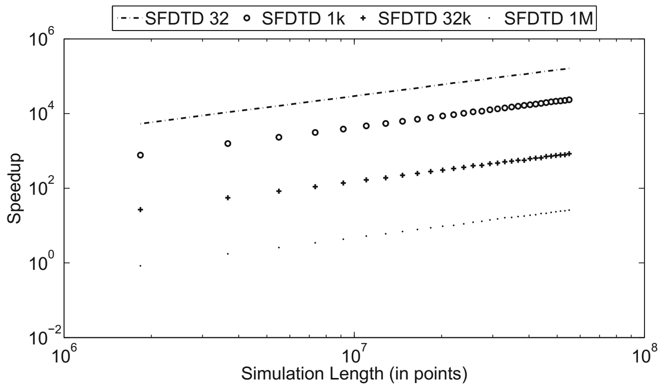


Fig. 6. Speed-up factor of SFDTD compared against FDTD

for the kernel execution, $T_{storing}$ is the time for writing to the hard drive, and T_{comm} is the communication transfer time. As previously mentioned, $T_{storing}$ is the bottleneck, due to the off-chip memory to hard drive transfers. For example, in Figs. 2, 3, 4 the SFDTD with segment size of 1M shows a $T_{kernel} = 0.012\,\text{s}$, $T_{comm} \approx 0.049\,\text{s}$. which are insignificant compared to $T_{storing} = 1.036\,\text{s}$. Based on the results and applying a linear regression, T_{exec} can be expressed in terms of the segment size, ς. This is $T_{exec} = (0.0021\varsigma + 4.6521)\,\text{ms}$.

The total execution time can be observed in Fig. 5, and it is clear how the SFDTD presents a similar behaviour as observed before in Figs. 2, 3, 4. However, the only in the case where the segment size is large (1M) and nearly equal to the simulation length shows the FDTD to perform better. The impact in the speed-up caused by selecting the SFDTD algorithm and the segment size provides a one to five orders of magnitude growth, having its peak at the smaller segment sizes. This effect can be observed in Fig. 6.

5 Conclusion

The proposed SFDTD algorithm exploit the memory access mechanism, with different types of fast memories (registers and shared memory) found in CUDA architecture. This allows an efficient use of fine-grained parallelism nested on a coarse-grained parallelism. In this paper, it has been shown that regardless of the segment size, the iteration execution time remains constant for different simulation space size. Also it was shown that the bottleneck is located at storing to disk, followed by the communication transfer time. But still, the SFDTD execution time is relatively minute compared to that using the normal FDTD implementation. Future works and discussion will be focus to 2D and 3D implementations, in addition to the instruction efficiency within the kernel.

References

1. Berenger, J.P.: A perfectly matched layer for the absorption of electromagnetic waves. J. Comput. Phys. **114**(2), 185–200 (1994)
2. Courant, R., Friedrichs, K., Lewy, H.: On the partial difference equations of mathematical physics. IBM J. Res. Dev. **11**(2), 215–234 (1967)
3. Demir, V., Elsherbeni, A.: Utilization of CUDA-OpenGL interoperability to display electromagnetic fields calculated by FDTD. In: Computational Electromagnetics International Workshop (CEM), pp. 95–98 (2011)
4. Drumm, I.: Implementing the FDTD tutorial. University of Salford - EPSRC Summer School, Salford, UK. http://www.acoustics.salford.ac.uk/res/drumm/FDTD-FE/Implementing%20FDTD%20Tutorial.doc (2007)
5. Engquist, B., Majda, A.: Absorbing boundary conditions for the numerical simulation of waves. Math. Comput. **31**(139), 629–651 (1977)
6. Herlihy, M.P.: Impossibility and universality results for wait-free synchronization. In: Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing, PODC '88, pp. 276–290. ACM, New York (1988)
7. Holland, R.: THREDE: a free-field EMP coupling and scattering code. IEEE Trans. Nucl. Sci. **24**(6), 2416–2421 (1977)
8. Ireland, D., Tee, W.C., Bialkowski, M.: Accelerated biomedical simulations using the FDTD method and the CUDA architecture. In: Microwave Conference Proceedings (APMC), 2011 Asia-Pacific, pp. 70–73. IEEE (2011)
9. Ju, F., Xing, C.: A study of parallel FDTD for simulating complex antennas on a cluster system. In: Microwave Conference Proceedings, 2005. APMC 2005. Asia-Pacific Conference Proceedings, vol. 5, p.4 (2005)
10. Kirk, D.B., Wen-mei, W.H.: Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann, Boston (2010)
11. Kondylis, G.D., De Flaviis, F., Pottie, G.J., Itoh, T.: A memory-efficient formulation of the finite-difference time-domain method for the solution of Maxwell equations. IEEE Trans. Microwave Theor. Tech. **49**(7), 1310–1320 (2001)
12. Kowalczyk, K.: Boundary and medium modelling using compact finite difference schemes in simulations of room acoustics for audio and architectural design applications. Ph.D. thesis, Queen's University Belfast (2010)
13. Lindman, E.: Free pace boundary conditions of the time dependent wave equation. J. Comput. Phys. **18**(1), 66–78 (1975)

14. Lipták, B.G.: Instrument Engineers' Handbook: Process Control and Optimization, vol. 2. CRC Press, Boca Raton (2005)
15. Livesey, M., Stack, J., Costen, F., Nanri, T., Nakashima, N., Fujino, S.: Development of a CUDA implementation of the 3D FDTD method. IEEE Antennas Propag. Mag. **54**(5), 186–195 (2012)
16. Moore, T., Blaschak, J., Taflove, A., Kriegsmann, G.: Theory and application of radiation boundary operators. IEEE Trans. Antennas Propag. **36**(12), 1797–1812 (1988)
17. Mur, G.: Absorbing boundary conditions for the finite-difference approximation of the time-domain electromagnetic-field equations. IEEE Trans. Electromagn. Compat EMC **23**(4), 377–382 (1981)
18. Nagaoka, T., Watanabe, S.: Accelerating three-dimensional FDTD calculations on GPU clusters for electromagnetic field simulation. In: 2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pp. 5691–5694 (2012)
19. Schroeder, M., Rossing, T.D., Dunn, F., Hartmann, W.M., Campbell, D.M., Fletcher, N.H.: Springer Handbook of Acoustics, 1st edn. Springer, New York (2007)
20. Taflove, A.: Application of the finite-difference time-domain method to sinusoidal steady-state electromagnetic-penetration problems. IEEE Trans. Electromagn. Compat EMC **22**(3), 191–202 (1980)
21. Unno, M., Aono, S., Asai, H.: GPU-based massively parallel 3-D HIE-FDTD method for high-speed electromagnetic field simulation. IEEE Trans. Electromagn. Compat. **54**(4), 912–921 (2012)
22. Wu, Y., Lin, M., Wassell, I.: Path loss estimation in 3D environments using a modified 2D finite-difference time-domain technique. In: 2008 IET 7th International Conference on Computation in Electromagnetics, CEM 2008, pp. 98–99 (2008)
23. Wu, Y., Wassell, I.: Introduction to the segmented finite-difference time-domain method. IEEE Trans. Magn. **45**(3), 1364–1367 (2009)
24. Xu, M., Thulasiraman, P., Noghanian, S.: Microwave tomography for breast cancer detection on cell broadband engine processors. J. Parallel Distrib. Comput. **72**(9), 1106–1116 (2012)
25. Yee, K.: Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. IEEE Trans. Antennas Propag. **14**(3), 302–307 (1966)
26. Yu, W., Liu, Y., Su, Z., Hunag, N.T., Mittra, R.: A robust parallel conformal finite-difference time-domain processing package using the MPI library. IEEE Antennas Propag. Mag. **47**(3), 39–59 (2005)
27. Yuan, X., Borup, D., Wiskin, J., Berggren, M., Eidens, R., Johnson, S.: Formulation and validation of Berenger's PML absorbing boundary for the FDTD simulation of acoustic scattering. IEEE Trans. Ultrason. Ferroelectr. Freq. Control **44**(4), 816–822 (1997)
28. Zhang, C., Qin, Y., Zhao, G., Zhu, Y.: Two-dimensional FDTD simulation for the acoustic propagation in a piezoelectric superlattice. In: 2009 IEEE International Ultrasonics Symposium (IUS), pp. 2031–2032 (2009)

Optimization of an OpenCL-Based Multi-swarm PSO Algorithm on an APU

Wayne Franz(✉), Parimala Thulasiraman, and Ruppa K. Thulasiram

University of Manitoba, Winnipeg, Canada
{umfranzw,thulasir,tulsi}@cs.umanitoba.ca

Abstract. The multi-swarm particle swarm optimization (MPSO) algorithm incorporates multiple independent PSO swarms that cooperate by periodically exchanging information. In spite of its embarrassingly parallel nature, MPSO is memory bound, limiting its performance on data-parallel GPUs. Recently, heterogeneous multi-core architectures such as the AMD Accelerated Processing Unit (APU) have fused the CPU and GPU together on a single die, eliminating the traditional PCIe bottleneck between them. In this paper, we provide our experiences developing an OpenCL-based MPSO algorithm for the task scheduling problem on the APU architecture. We use the AMD A8-3530MX APU that packs four x86 computing cores and 80 four-way processing elements. We make effective use of hardware features such as the hierarchical memory structure on the APU, the four-way very long instruction word (VLIW) feature for vectorization, and global-to-local memory DMA transfers. We observe a 29 % decrease in overall execution time over our baseline implementation.

Keywords: PSO · Parallel evolutionary computing · APU · OpenCL

1 Introduction

As we move into the multi-core era, parallel systems are becoming increasingly heterogeneous, incorporating everything from multi-core CPUs to accelerators, GPUs, and FPGAs. This complicates the task of scheduling a parallel workload.

Formally, the *task matching* problem seeks to map a set of tasks T (of differing lengths) to a set of distributed machines M (with varying capabilities) such that the overall time required to complete all tasks (*makespan*) is minimized. Task matching has been well-studied, and is known to be NP-complete [2]. Therefore, heuristics such as Particle Swarm Optimization (PSO) [3] are often considered. PSO simulates a group (or *swarm*) of particles as they move and interact within a solution space. Each *particle* encodes a set of inputs to an optimization problem via its coordinate values. PSO imposes forces to push particles toward more promising areas of the solution space.

We consider the multi-swarm PSO (MPSO) implemented by Solomon *et al.* [6]. This work applied MPSO to task matching using CUDA on a discrete GPU.

While these types of approaches present good discussions of thread-to-data mapping strategies and impressive speedup results, further optimizations are achievable by taking simple efficiency metrics like device utilization into account.

Recently, AMD introduced a new architecture that combines a CPU and GPU together on a single die (AMD calls this an Accelerated Processing Unit, or APU). In this paper, we provide our experiences developing and optimizing MPSO in OpenCL on the APU architecture.

2 Architecture and Runtime System

The Open Computing Language (OpenCL) is a parallel language and runtime system developed for heterogeneous devices. OpenCL refers to threads as *work-items*. Work-items can be combined into *workgroups* to synchronize or share information. The host (CPU) launches *kernels* (functions that execute on the GPU), transferring any required data from host to device memory.

Our on-die GPU [1] consists of five *compute units* (CUs), each with 16 *Processing Elements* (PEs). Each PE is a very large instruction word (VLIW) processor with four arithmetic logic units (ALUs). We use OpenCL vector data types of size four to match this arrangement.

AMD hardware runs work-items in groups of 64 called *wavefronts*. Each CU executes one wavefront at a time. However, CUs can maintain the states of up to 32 wavefronts (*active wavefronts*). During global memory accesses, another wavefront may be swapped in and executed to hide latency.

GPUs provide three main types of memory. *Global memory* is large (512 MB), but has a high latency. To accommodate the VLIW layout of the PEs, the memory bus is 128 bits wide. A feature called *coalescing* provides enhanced performance when consecutive work-items access contiguous chunks of memory.

Constant memory is a read-only store (64 KB) that provides a broadcast function. When multiple work-items access the same location, data is read once and broadcast to all work-items.

Local memory is a small (32 KB/CU), fast store. When each work-item in a quarter wavefront accesses consecutive chunks of 8 bytes, all memory banks are active. If the access pattern differs, multiple work-items will access the same memory bank (a *bank conflict*), and it will service the requests sequentially.

3 Algorithm, Implementation and Optimization

This section describes the MPSO algorithm, the implementation of each of the kernels in this algorithm, and their optimization. We compare *vectorized* (using four-element vector data types) kernels, *unvectorized* (using scalar data types) kernels, and optimized variants.

We define the following symbols: n, the user-defined maximum number of iterations; s, the number of swarms; p, the number of particles per swarm; $d = |T|$, the number of dimensions; $m = |M|$, the number of machines; and e, the number of particles exchanged between swarms at each interval.

Our GPU only allows one kernel to be executed at once. This makes it important to ensure that *kernel occupancy* (device utilization) is as high as possible. Kernel occupancy [1] is the ratio of the number of active wavefronts per CU to the maximum number of wavefronts supported per CU. *Estimated Kernel Occupancy* (EKO) refers to occupancy as measured by AMD profiling tools.

3.1 MPSO Algorithm

In PSO, each particle k maintains three pieces of information: a position vector X_k, a personal-best position vector $\acute{X}_k$, and a velocity vector V_k. Each swarm j also maintains the position of the best solution seen so far by any particle, $\hat{X}_j$. On each iteration i, velocity is modified according to the following equation [5]:

$$V_k^{i+1} = \omega * V_k^i + c_1 * R_1 * (\acute{X}_k - X_k^i) + c_2 * R_2 * (\hat{X}_j - X_k^i) \tag{1}$$

Position is then updated using:

$$X_k^{i+1} = X_k^i + V_k^{i+1} \tag{2}$$

R_1 and R_2 are vectors of random numbers between 0 and 1 (selected from a uniform distribution), while c_1, c_2, and ω are user-defined weighting parameters used to balance the forces exerted by each of the three terms.

In our solution space, each dimension represents a task, while each coordinate value along a dimension represents a machine that task may map to. Our MPSO algorithm arranges independent swarms in a ring topology. Every given number of iterations, the e best particles from each swarm replace the e worst particles of its neighbour. Ranking is performed using an objective function, which accepts a position X_k, and returns a fitness f_k, a measure of solution optimality. To prevent redundant calculations, we also store the current and personal-best fitnesses (f_k^i and $\acute{f}_k^i$, respectively) for each particle, and the swarm-best fitness $\hat{f}_j$ for each swarm. An outline of sequential MPSO is shown in Algorithm 1.

Optimizations below are illustrated using $s = 60$ and $p = 128$ for a total of $n = 1000$ iterations. For other PSO parameters, we set $d = 80$, $m = 8$, $e = 24$, $c1 = 2.0$, $c2 = 1.4$, and $\omega = 1.0$. We exchange particles every 10 iterations. Execution time is averaged over 30 trials to account for stochastic variation.

3.2 Data Layout

We employ a lookup table called an *Estimated Time to Complete* (ETC) matrix [6] to prevent redundant makespan calculations. An ETC value at row i, column j records the time that machine i requires to execute task j. We generate the matrix using the host (CPU) while the initial random number generation and particle initialization kernels are running on the device.

Particle data is stored in global memory arrays. Values are ordered by dimension (in groups of four for VLIW performance), then particle index. This ensures coalescing within kernels that map one work-item to each particle.

Algorithm 1. Sequential MPSO Algorithm

```
 1: Initialize particle positions and velocities randomly
 2: for iteration $i = 0$ to $n - 1$ do
 3:    for swarm $j = 0$ to $s - 1$ do
 4:       for particle $k = 0$ to $p - 1$ do
 5:          Calculate $f_k^i$ from $X_k^i$
 6:          Update $V_k^i$ using (1)
 7:          Update $P_k^i$ using (2)
 8:          Update best values ($\acute{X}_k^i$, $\hat{X}_j$, $\acute{f}_k^i$, $\hat{f}_k$) if necessary
 9:          if $i$ mod $numSwapIters = 0$ then
10:             Overwrite $e$ worst particles in next swarm with $e$ best in this swarm
11:          end if
12:       end for
13:    end for
14: end for
```

3.3 Random Number Generation

MPSO requires a large number of random values. Salmon *et al.* [4] recently proposed a counter-based parallel pseudo random number generator that is well-suited for the GPU. The only state storage requirements are simple counter values that are incremented before each call. We keep a large (142 MB) buffer of random numbers in global memory, which is refilled on demand. Our completed kernel consumes 8 registers per CU, yielding an 87.5 % EKO.

3.4 Particle Initialization

This kernel uses one work-item per particle dimension ($s * p * d$ work-items) to initialize X and V in the ranges $[0, m - 1]$ and $[0, 2m]$, respectively. In addition, $\acute{f}$ and $\hat{f}$ are set so as to ensure an update on the first fitness calculation. Since the work-item-to-data mapping is one-to-one, it is easy to use four-way vector data types, reducing the number of required work-items by a factor of four.

Register usage is very low, since few calculations are performed. This is desirable because there is a pool of registers allocated to each CU. Therefore if kernel instances require many registers, fewer wavefronts can be run simultaneously on the CU [1]. This kernel achieves 100 % EKO.

Table 1 shows the average run time for vectorized and unvectorized versions of this kernel. The table shows two effects of vectorization. First, it significantly decreases the percentage of time that the fetch unit is busy, as it generates only one quarter of the memory requests.

Second, vectorization increases the *ALU-Op-to-Fetch-Op Ratio* slightly. This ratio can be used to determine the memory-boundedness of a kernel. If it is low, the kernel is more memory-bound. However if it is high, the kernel is more compute-bound, increasing the effectiveness of wavefront context switching.

In this case, the small difference is due to an increase in the number of calculations performed per work-item. However, 9.03 ALU ops is far too few to cover the latency period of 300–600 cycles incurred by a single global memory fetch. In spite of the memory system's ability to keep up with our requests, and 100% EKO, execution time is still limited by memory-boundedness here.

3.5 Update Fitness

This kernel is our objective function. We map one work-item to each particle k to calculate f_k, then write this value to global memory. To do this, we use a local memory buffer of size m for each work-item. Work-items iterate through their particle's dimensions (tasks), and add the time required for the chosen machine (axis value) to the local array. This time is retrieved from the ETC matrix.

This kernel requires a large number of registers, severely limiting the number of wavefronts we can launch, and resulting in an EKO of only 50 %. The low number of active wavefronts will also limit opportunities for context switching.

In kernels that use local memory, it is common [6] to assign each swarm to one workgroup. We relax this constraint and incorporate as many swarms into one workgroup as possible. Since the execution of each workgroup is limited to one CU, allocating larger workgroups increases device utilization.

The makespan computation incurs a number of irregular local memory accesses and bank conflicts because we do not know in advance which machine each task will map to. The most achievable approach for vectorization is to cause each work-item to handle a vector of four dimensions (tasks) at once. However, the values along these dimensions may not refer to consecutive machines. This means that on each iteration, each work-item must make four scalar ETC accesses (rather than one four-element access), and four atomic writes to local memory (as multiple tasks could map to the same machine).

One avenue for optimization is to "cache" the four previously read local memory values in registers. If subsequent tasks map to the same machine, we can skip a read, grabbing the data from the register. However, this introduces a number of conditional statements, which are a bottleneck for data-parallel hardware. Applying this technique actually resulted in a small increase (0.11 ms) in average kernel execution time.

3.6 Update Bests

In this kernel, we update the particle-best and swarm-best fitnesses and positions, launching one work-item per particle. The kernel operates in two phases.

The first involves a simple comparison between $\acute{f}_k^{i-1}$ and f_k^i for each particle k. If an update is required, each work-item overwrites $\acute{f}_k$ and $\acute{X}_k^i$ in their global memory arrays. The second phase performs a parallel reduction to find $\hat{f}_j$ for each swarm j and update it, if necessary. Finally, work-items cooperate to update the $\hat{X}_j$, if necessary. There is a relationship between these two phases that we can exploit: $\hat{f}_j$ will only ever be set to a newly updated $\acute{f}_k$.

The second phase requires local memory buffers for the parallel reduction. We can increase the efficiency of our code by re-using these buffers during the first phase to store information about which particles' best values were updated. This information will allow us to exploit the relationship mentioned above.

In phase 1, we perform the comparison and write the boolean result to local memory. Phase 2 reuses the local buffer to perform a reduction, skipping over any unneeded values. Finally, $\hat{f}_j$ and $\hat{X}_j$ are updated in global memory, if necessary.

As we move from the unvectorized to the vectorized kernel, register usage forces a 25 % decrease in EKO (see Table 2). However, this is outweighed by the increase in the ALU-to-fetch ratio that vectorization brings.

Table 1. Particle initialization (T: Avg. time, FB: Fetch unit busy)

Kernel	T (ms)	ALU:Fetch	FB (%)
unvec	0.988	7.52	22.65
vec	0.980	9.03	6.85

Table 2. Update bests (T: Avg. time, R: Registers)

Kernel	T (ms)	R	EKO (%)	ALU:Fetch
unvec	0.224	6	100	76.5
vec	0.081	9	75	194.5

3.7 Update Position/Velocity

In this kernel, we update each particle's current velocity and position. Since Eq. (1) allows each dimension to be calculated independently, we can launch a full $s * p * d$ work-items. Note that in Eq. (1), the same swarm-best positions ($\hat{X}_j$) are read by all $p * d$ work-items in a swam j. Therefore, we place them in constant memory to take advantage of broadcasting.

Table 3 shows the effect of vectorization (with each work-item handling four dimensions). EKO is limited to 75 % due to the number of registers required for the calculation. One option to reduce register usage is to break the kernel into multiple pieces, one for each of the terms in Eq. (1). However, each kernel would have to store/reload partial results using global memory, and there are not enough ALU operations in each term to cover the resulting latency.

A second option is to attempt to use local memory in place of one or more registers. OpenCL provides a function that asynchronously copies data from global to local memory (via a DMA transfer) within a kernel. We were able to replace a single register using this call. As Table 3 shows, this was enough to raise the EKO. However, overall MPSO execution time also rises, as local memory's bandwidth is lower than that of registers. To support four-way vectorization, each work-item must read four (rather than the optimal two) values from local memory, causing bank conflicts, and further increasing execution time.

A third option involves combining this kernel with an adjacently launched kernel. This should allow the compiler to re-use registers between what was previously separated code. The update bests kernel is launched immediately before this kernel, and is our only practical candidate for a merger (the following kernel is only invoked on iterations when particle swapping occurs).

One problem here is that the update bests kernel requires $s * p$ work-items, while the update position/velocity kernel requires $s*p*d$ work-items. Ordinarily,

Table 3. Update position/velocity

Kernel	Avg. time (ms)	Registers	EKO (%)	MPSO time (s)
unvec	1.209	6	75	2.749
vec	1.005	9	75	1.981
vec (async copy)	1.127	8	87.5	2.087
vec (combined)	1.177	10	75	2.124

we could simply launch the maximum number of required work-items and restrict the parallelism for the update bests code. However, this is complicated by the fact that the kernels require different work-item-to-data mapping strategies.

Each work-item in the update position/velocity kernel operates independently. As a result, the kernel may split the work for one swarm across multiple workgroups. Work-items performing the update bests kernel must cooperate using local memory. Since local memory cannot be used for inter-workgroup communication (only intra-workgroup), all work-items operating on a swarm must be members of the same workgroup.

With this in mind, we launch the combined kernel using $s * p$ work-items, and implement a loop that iterates through the update position/velocity kernel code d times. While this will increase the individual execution time of a kernel instance, it has little effect on occupancy.

Table 3 shows the result of this strategy. The register usage of the combined kernel is less than the sum of usage counts of the individual kernels. Unfortunately, this is not enough to increase EKO. The loop pushes the combined kernel execution time slightly above the combined duration of the original kernels.

3.8 Find Best/Worst Particles

In this kernel, we must determine the indices of the particles with the e best and e worst fitnesses in each swarm. This information is stored in global memory buffers so that the exchange can be done by a later kernel.

We begin by mapping one work-item to each particle, employing an algorithm used by Solomon *et. al.* [6]. This involves copying fitness data to local memory and performing e parallel reductions. Tracking the particle-indices of these particles requires additional local buffers. In total, five buffers of size p are used.

Four-way vectorization will quadruple the local memory requirements per CU. Instead, we use two-way vectorization, which also provides a more efficient access pattern for local memory. Table 4 shows statistics for this kernel. Local memory usage drops the EKO to only 25 %. With this in mind, we consider an alternative algorithm requiring less local memory.

Our new algorithm performs an all-to-all comparison, recording the number of times each element wins. After this process, the ids of the work-items with the highest (lowest) e win counts are the indices of the e best (worst) particles.

This approach is complicated by the fact that particles can have identical fitnesses. We use a race condition (in a loop) to detect this case, by forcing all work-items to write their id to a local memory location corresponding to their win count. Values are then read back. If a work-item finds an unexpected value, it increments its win count by one. Otherwise it is masked out of the loop.

This loop stops when all work-items read back their own ids. Using a separate local memory variable to record the loop's stopping condition pushes our local memory usage over the top, limiting occupancy. Instead, we temporarily re-use an existing local memory buffer element on each iteration.

This approach uses only p local memory space, removing the cap on EKO.

Table 4. Find best/worst (T: Avg. time, M: Local memory usage)

Kernel	T (ms)	M (bytes)	EKO (%)
unvec (initial)	1.850	6144	62.5
vec (initial)	1.658	12288	25
vec (alt)	0.269	2048	100

Table 5. Swap particles (T: Avg. time, R: Registers)

Kernel	T (ms)	R	EKO (%)
unvec	2.463	7	100
vec	0.048	20	37.5
vec (shared)	0.646	8	87.5

3.9 Swap Particles

This kernel performs the actual particle exchange between swarms. Here we launch one work-item for each dimension of every particle to be exchanged ($s{*}e{*}d$ work-items). The e best particles in each swarm overwrite the e worst particles in the next swarm. Specifically, X, V, $\hat{X}$, and $\hat{f}$ are overwritten. Current fitnesses will be recalculated on the next iteration before they are needed.

This kernel makes a large number of global memory accesses. A vectorized algorithm makes fewer, larger accesses, reducing the average time (see Table 5). However, a large number of index calculations are needed to move the array values between swarms. Vectorization drives up register usage, limiting EKO.

We could attempt to replace registers that deal with best and worst particle indices with local memory. One limitation of the asynchronous copy function is that it must be called with the same arguments for all work-items in a work-group. For the best indices, this is not a problem. But for the worst indices, we must retrieve the next swarm's value (an offset of one). This means that there will be wrap-around in the global memory array we are reading from. Since we are handling multiple swarms per workgroup, this leads to different values for different work-items in the same workgroup. We therefore manually copy the worst indices to local memory using a loop.

The asynchronous copy call also forces us to move the best and worst indices from constant memory to global memory (OpenCL provides no means of copying from constant to local memory asynchronously). This seems to allow the compiler to significantly reduce register usage. We speculate that as constant memory is read-only, the compiler uses some extra registers to store the retrieved values.

Table 5 shows that this increases EKO significantly. Unfortunately, the decrease in bandwidth when moving to local memory, coupled with inability of global memory to broadcast, still results in a larger kernel execution time.

4 Results

Our test system uses an AMD A8-3530MX APU. This device incorporates a quad core CPU at 1.90 GHz with 6 GB of RAM, and an on-die Radeon HD 6620G graphics processor. All experiments were compiled with Visual Studio 2010 using the AMD APP SDK v2.7 implementation of OpenCL 1.2. Hardware performance counters were measured using AMD CodeXL software.

Optimizations that alter kernel occupancy tend to perform better with large input sizes than small ones. We present the results of a simple scaling experiment below. We fix s at 60 and scale up p in the range $[4, 256]$. The other MPSO parameters remain as presented in Sect. 3.1. We run each instance 30 times and average execution time to account for stochastic variance.

Figure 1 shows the effect that scaling p has on execution time. The gap between unvectorized and vectorized algorithms widens as p increases. This is true in spite of the fact that unvectorized kernels generally have a higher occupancy level. This trend reveals multiple levels of parallelism at work. In addition to parallelism at the work-item level (corresponding to kernel occupancy), there is also parallelism at the ALU level (corresponding to vectorization). In this case, the latter outweighs the former and we see the widening gap in the graph.

The slight rises and falls in the slopes of the lines result from combining multiple swarms into the same workgroup. In cases where the mapping works out evenly, we see a lower execution time due to an increase in parallel efficiency. But if an extra swarm narrowly fails to fit into a workgroup, the hardware scheduler my need to launch an extra round of workgroups on the CUs.

The update fitness caching optimization exhibits no improvement over the vectorized algorithm. Combining the update position/velocity and update bests kernels ultimately results in a higher execution time than the plain vectorized algorithm. This line does not quickly diverge from the vectorized series because the EKO of the combined kernel remains identical to both separate versions. However, the combined kernel suffers from lower parallelism at the work-item level, since it must iterate for the position/velocity section.

Moving the position/velocity kernel to local memory significantly increases execution time (though it increased occupancy) due to the lower bandwidth of local memory. On the other hand, moving the swap kernel to local memory has only a small effect. We attribute this to the fact that the local swap kernel uses much less local memory than the local position/velocity kernel. Finally, our alternative find best/worst algorithm decreases execution time slightly, in a manner relatively proportional to swarm size.

Our largest input size is the rightmost point in Fig. 1. Moving from the unvectorized to the best-performing algorithm (find best/worst alt), we see a decrease of 1.51 s, or approximately a 29 % improvement.

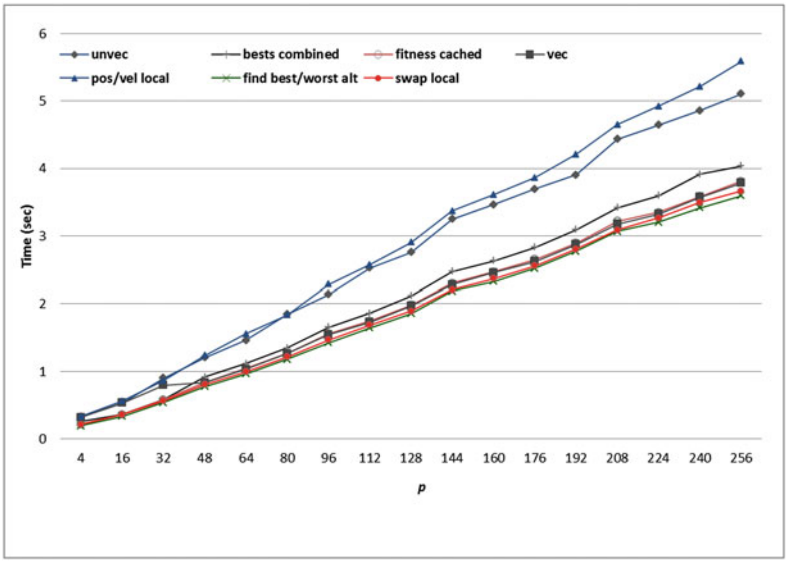


Fig. 1. Execution time as the number of particles per swarm increases.

5 Conclusion

This work has traced the optimization process of an MPSO algorithm on an APU architecture. We have investigated optimizations to improve kernel occupancy and vectorization, optimize memory access patterns, and reduce register usage.

Many of the ideas presented here can be directly applied to other parallel population-based algorithms. It is worth noting that even those ideas that did not consistently provide benefit to us may prove worthwhile in other contexts or on other devices.

Funding for this work was provided in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and the University of Manitoba.

References

1. Advanced Micro Devices: AMD Accelerated Parallel Processing OpenCL Programming Guide. http://developer.amd.com/download/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide.pdf (2012)
2. Fernandez-Baca, D.: Allocating modules to processors in a distributed system. IEEE Trans. Softw. Eng. **15**(11), 1427–1436 (1989)
3. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: IEEE International Conference on Neural Networks, Perth, Western Australia, vol. 4, pp. 1942–1948 (1995)
4. Salmon, J.K., Moraes, M.A., Dror, R.O., Shaw, D.E.: Parallel random numbers: as easy as 1, 2, 3. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, Seattle, WA, USA, November 2011, pp. 16:1–16:12. http://doi.acm.org/10.1145/2063384.2063405 (2011)

5. Shi, Y., Eberhart, R.: A modified particle swarm optimizer. In: Proceedings of the Evolutionary Computation, IEEE World Congress on Computational Intelligence, Anchorage, AK, USA, May 1998, pp. 69–73 (1998)
6. Solomon, S., Thulasiraman, P., Thulasiram, R.: Collaborative multi-swarm PSO for task matching using graphics processing units. In: ACM Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO), Dublin, Ireland, July 2011, pp. 1563–1570. http://doi.acm.org.proxy1.lib.umanitoba.ca/10.1145/2001576.2001787 (2011)

Core Allocation Policies on Multicore Platforms to Accelerate Forest Fire Spread Predictions

Tomàs Artés(✉), Andrés Cencerrado, Ana Cortés, and Tomàs Margalef

Computer Architecture and Operating Systems Department,
Universitat Autònoma de Barcelona, Campus UAB, Edifici Q,
08193 Bellaterra, Spain
{tomas.artes,andres.cencerrado,ana.cortes,tomas.margalef}@uab.cat
http://caos.uab.es

Abstract. Software simulators are developed to predict forest fire spread. Such simulators require several input parameters which usually are difficult to know accurately. The input data uncertainty can provoke a mismatch between the predicted forest fire spread and the actual evolution. To overcome this uncertainty a two stage prediction methodology is used. In the first stage a genetic algorithm is applied to find the input parameter set that best reproduces actual fire evolution. Afterwards, the prediction is carried out using the calibrated input parameter set. This method improves the prediction error, but increments the execution time in a context with hard time constraints. A new approach to speed up the two stage prediction methodology by exploiting multicore architectures is proposed. A hybrid MPI-OpenMP application has been developed and different allocation policies have been tested to accelerate the forest fire prediction with an efficient use of the available resources.

Keywords: Forest fire · Simulation · Data uncertainty · Hybrid MPI-OpenMP · Evolutionary computation · Resource assignment · Multicore architecture

1 Introduction

Some natural hazards involve serious consequences from the environmental, economic and social point of view. Therefore, it is critical to react as soon as possible to minimize their effects. This work focuses on forest fire spread prediction. This kind of natural disasters are among the most worrisome in southern European countries. To deal with this hazard, models which describe the forest fire spread have been developed and implemented in simulators. In the case of forest fire, the Rothermel model [1] is one of the most used and proven. FARSITE [2] is a widely used simulator which implements this model. To perform a simulation, it requires a set of parameters that describes the environment where the fire is taking place. These input parameters may present several difficulties: some of them are not uniform along the forest fire scenario, others can present a temporal variability, others must be estimated by interpolated measures, others must be obtained from complementary models. This fact results in a certain degree of uncertainty in input data that provokes a lack of accuracy in the prediction. A calibration method has been used to reduce the uncertainty of the input data set [3]. In this method the prediction is based on two stages where firstly a calibration is carried out and afterwards the prediction is done. The first stage consists on searching the input parameters set that best reproduces the actual fire behavior. This search is carried out applying a Genetic Algorithm (GA) [4] over a certain number of iterations. Once the preset number of iterations is reached, the best set of parameters is used to carry out the prediction for the following time step. Although this prediction approach provides better results in terms of quality degree, it is more computing demanding. The increase in the time needed to obtain a satisfactory prediction result could not always be a feasible approach when dealing with emergencies. For this reason, the calibration stage has been implemented using a message passing Master/Worker paradigm to take advantage of parallel/distributed systems. The two stage prediction scheme was originally designed to run each fire spread simulation using a single core. In previous studies, it has been stated that every fire spread simulation lasts shorter or longer depending on the particular setting of the input parameters. Actually, the time needed to complete every generation of the calibration stage is bounded by the worker that lasts longer. Therefore, the just mentioned master/worker scheme using a single core per worker could eventually generate unbalance among workers. For this reason, the kernel of the used simulator (FARSITE) has been parallelized using OpenMP in order to reduce execution time. In this context, it is not worthy dedicating the same amount of resources to short and long simulations. Therefore, a methodology to characterize FARSITE which allows us to assess, beforehand, the execution time of a given scenario has been developed. This ability enables the possibility of designing core allocation policies to minimize the total execution time of the prediction scheme and, to use the available resources more efficiently. In Sect. 2, the two stage methodology and its implementation are described. Subsequently, in Subsect. 2.1 the results of the kernel parallelization are detailed. In Sect. 3, the methodology used to detect slower kernel executions is briefly presented. The results of the hybrid MPI-OpenMP with different core allocation policies are shown in Sect. 4. Finally, conclusions and future work are described in Sect. 5.

2 Hybrid MPI-OpenMP Master/Worker Prediction Scheme

A simulator independent data-driven prediction scheme is used to calibrate the input data set provided to the simulator [3]. For this purpose, a previous calibration step is introduced, as can be seen in Fig. 1. So, the input data set used for the prediction stage is calibrated in this first stage for each prediction step. Based on the hypothesis that the meteorological conditions will not suddenly change from the calibration stage to the prediction stage, the calibrated data set could be used to produce a more accurate prediction.

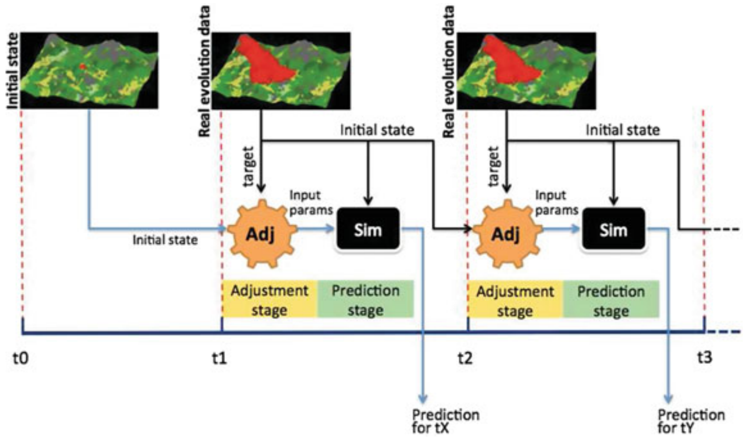


Fig. 1. 2 stage prediction method

Because of their outstanding results within this framework [5] this work is based on the use of GA as calibration technique. The algorithm starts from an initial random population of individuals, each one representing a scenario to be simulated. An individual is composed of a number of different genes that represent input variables such as dead fuel moistures, live fuel moistures, wind speed and direction, among others. Each individual is simulated and it is evaluated comparing the predicted and real fire propagation by estimating the fitness function described in Eq. 1. This fitness function computes *the symmetric difference between predicted and real burned areas.*

$$Difference = \frac{UnionCells - IntersectionCells}{RealCells - InitCells} \tag{1}$$

In Eq. 1, *UnionCells* is the number of cells which describe the surface burned considering predicted fire and the real fire. *IntersectionCells* is the number of cells burned in the real map and also in the predicted map, and *RealCells* are the cells burned in the real map. *InitCells* is the number of cells burned at the starting time. This difference takes into account the wrong burned cells and the mistaken for burned cells. According to this fitness function the whole population is ranked and the genetic operators *selection*, *elitism*, *mutation* and *crossover* are performed over the population, producing an evolved population which will have, at least, the best individual of the last generation (elitism). The new population is then evaluated in the same way. This iterative process allows us to find a good input parameter set, but it involves high computational cost due to the large amount of simulations required. Therefore, it is essential to speed up the execution keeping the accuracy of the prediction. For this reason, an implementation of the two stage methodology has been developed using High Performance Computing techniques. Since the GA fits the Master/Worker paradigm, an MPI implementation has been developed. At the first stage, the master node generates an initial random population which is distributed among the workers. Then, the workers simulate each individual and evaluate the fitness function. The errors generated by the workers are sent to the master which sorts the corresponding individuals by their error before applying the genetic operators and producing a new population. This iterative process is repeated a fixed number of times. The last iteration (generation) contains a population from which the best individual is taken as the best solution, and then it is used in the prediction stage. Since every simulation can be carried out in a parallel way, the individual whose simulation takes longer determines the elapsed time for that particular generation. In order to shorten simulation times, FARSITE has been analyzed with profiling tools such as OmpP [6] and gprof [7] to determine which regions of the code could be parallelized with OpenMP. The result of such analysis determined the particular loops that could be parallelized using OpenMP pragmas. The results of such parallelization have been presented in [8]. The parallelized loops represents about 60 % of one iteration execution time. It means that 40 % of the iteration execution time is sequential and it implies that the speed up is not linear, but is limited by such sequential part.

2.1 Evaluating the Hybrid Scheme

In order to have an snap shot of the Hybrid Master/Worker scheme potential, we performed two preliminary test experiments. For this purpose, we used as a terrain, a geographical zone of high risk of forest fire located in the North-East of Spain, *Cap de Creus.* In this terrain, a synthetic fire has been simulated using as input setting information provided by the local meteorological centre SMC (Catalan Meteorological Service) obtained from the Automatic Weather Station Network (XEMA). The vegetation types has been obtained from CORINE land cover data base [9]. The obtained fire spread was used as a real fire for comparison purposes. The computational platform used was a 32 nodes IBM x3550 cluster where each computing node has two dual core Intel Xeon 5160 and 12 GB of memory at 667 Mhz. The first experiment consists of executing the hybrid Master/Worker prediction scheme using a random initial population of 25 individuals, which was evolved 10 generations using one core for each individual. The resulting evolved populations obtained at each iteration of the GA, are recorded and reused again but using 4 core per individual. This way, both cases use the same individuals at each iteration and they only differ in the cores assigned to each individual. Figure 2 depicts the calibration errors evolution through obtained over the evolution of the population and shows the elapsed time to execute the 25 individuals of each iteration. As it was expected, the execution time is reduced significantly when the number of cores is increased. For example, the total execution time is reduced from 72693 s to 27174 s. The error evolution is consistent because the error for the next generation must be equal, as least, to the last generation. This is due to the elitism genetic operator. The best individual is introduced without changes to the next population to evaluate. If crossover or mutation operator does not create a new individual better than the individual chosen by elitism, there are no error improvement. This fact produces a stair like behaviour when plotting the error for each generation. Bearing in mind the results obtained in this preliminary study, to increase the number of cores assigned to FARSITE simulations (individuals of the GA algorithm) could not always be a benefit in terms of final execution time. In particular, for those short simulations (several seconds) that will not bound the duration of a given generation, it will not be useful to allocate more than one core. Therefore, in order to be able to determine how many cores to assign to each FARSITE simulation (individual evaluation in the GA scheme), it is necessary to be able to asses before running the simulation, its execution time. In the next section, we describe a methodology to characterize FARSITE that allows us determining in advance the duration of a given FARSITE simulation for a certain input settings. In particular, the estimated execution time is defined by and interval time called class.

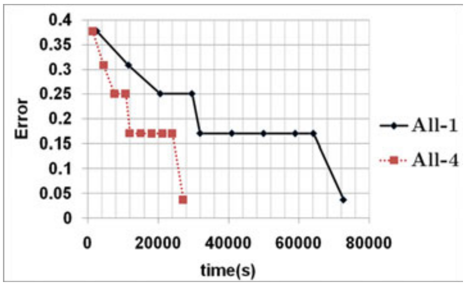


Fig. 2. Execution time and error considering 1 and 4 cores

3 FARSITE Characterization

As it has been mentioned, the execution time of a single simulation on the same map and simulating the same time can vary from seconds to several minutes or even hours depending on the input settings of the fire spread simulator. Consequently, in order to design core allocation strategies, we need to be able to anticipate the execution time of a certain input setting without the necessity of running that simulation. For this purpose, we propose a real time strategy to rapidly assess for a given input scenario (simulator input setting) an execution interval time where the corresponding FARSITE simulation time will fit. This strategy has been successfully tested for several forest fire spread simulators as is reported in [10]. To be operative during a real hazard, this execution-time estimation of a given scenario must be inferred as quickly as possible, keeping the cost of carrying out this operation to a minimum, in terms of time needed. For this reason, we rely on the field of Artificial Intelligence to be able to automatically learn from stored knowledge, so as to provide smart decisions. In particular, we use Decision Trees to extract this knowledge from certain database. The FARSITE characterization (or simulator kernel characterization) is fulfilled by means of carrying out large sets of executions (on the order of thousands) counting on different initial scenarios (different input data sets), and then, applying knowledge-extraction techniques from the information they provide, i.e. we record the execution times from the experiment, and then we establish a classification of the input parameters according to the elapsed times they produced. Specifically, we follow this sequence of steps:

– *Training database building*: Currently, we work with training databases composed of 12000 different scenarios.
– *Determination of execution time classes*: The whole training database is executed, and every par [scenario, execution time] is recorded. After this, the histogram of execution times is analyzed. Identifying the local minimums of the histogram the upper and lower boundaries of each *execution time class* are determined. Figure 3 depicts the histogram of execution times obtained for FARSITE. From the analysis of this histogram the resulting classes are the following:

  • Class A: ET $\leq$ 270 s.
  • Class B: 270 seconds $<$ ET $\leq$ 750.
  • Class C: 750 seconds $<$ ET $\leq$ 1470 s.
  • Class D: 1470 seconds $<$ ET $\leq$ 3600 s.

– *Decision Tree building*: once we have determined how many classes we will consider, then we are ready to build the Decision Tree. For this purpose, we rely on the C4.5 algorithm, specifically, the J48 open source Java implemented in the Weka data mining tool [11].

These steps are carried out off-line, in a preparation phase before any real emergency is under analysis. Once this methodology has been followed, only one final step remains: the application of the built Decision Tree with the scenario describing the ongoing fire, in order to assess in advance the execution time its simulation will produce. This action supposes a negligible cost, in terms of time overhead (on the order of a few seconds) and enables the ability of deciding at real-time how many cores allocate to each simulation depending on its estimated execution time class. The results of applying Decision Trees were validated in [12], where it is demonstrated that we reach 96.4 % correct classifications. In the following section, we reported an experimental study where different core allocation strategies have been analyzed in terms of speedup an efficiency.

4 Experimental Study

By means of the simulator kernel Characterization described in Sect. 3, we are able to identify those individuals that will last longer in a given iteration of the Hybrid Master/Worker prediction scheme. This fact allows us to take advantage of the OpenMP parallel version of FARSITE, determining a real-time core-allocation strategy for each individual in a generation in order to save as much

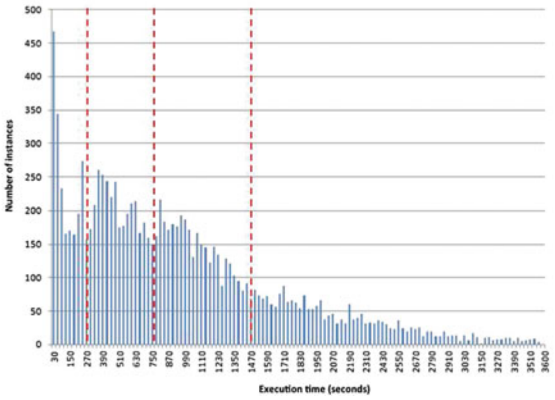
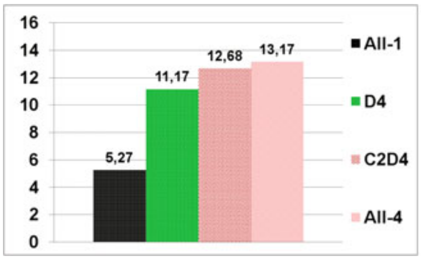


Fig. 3. Histogram of execution times using FARSITE. vertical dotted lines indicate the defined classification boundaries.

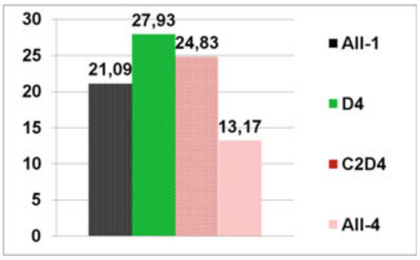
overall execution time as possible. Thus, in this section, the benefits of applying the FARSITE characterization together with the developed Hybrid Master/Worker prediction scheme are evaluated. For this purpose, we conducted an experimental study consisting of applying different core-allocation policies and analyzing the obtained speedup and efficiency. The execution platform and the terrain where the experiments have been performed are the same that the ones used in Sect. 2. The experiments carried out consist of executing the Hybrid Master/Worker prediction scheme using 9 different initial populations each one composed of 25 individuals and the GA iterates 10 generations. As it was done in the preliminary study reported in Sect. 2.1, the intermediate populations obtained at each generations for the case of one single core allocation per simulations, have been recorder to be able to reproduce the same evolution with different allocation policies which are listed following:

– *All-1*: The basic case, where each individual is allocated to one core.
– *C2D4*: Individuals belonging to class C are allocated to 2 cores. Individuals belonging to class D are allocated to 4 cores. The rest of individuals are allocated to one core.
– *D4*: Individuals belonging to class D are allocated to 4 cores. The rest of individuals are allocated to one core.
– *All-4*: All the individuals are allocated to 4 cores. It is worth mentioning that, for this experimental study, the workers are limited to execute only an individual per generation. This fact reduces the efficiency because parts of the workers are waiting for the slowest simulation to end.

It is worth mentioning that, for this experimental study, the workers are limited to execute only an individual per generation. This fact reduces the efficiency because some workers are waiting for the slowest simulation to end but it ensures seeing the effects of the different core-allocations. Figure 4(a) shows the results obtained. This figure presents the average speedup (sequential execution time/parallel execution time) obtained by applying the 4 different allocation policies. As it was expected, policy *All-4* was the most favorable in terms of speedup and it is useful to compare to the speedup reached by the other policies. It can also be observed that policy *C2D4* is the most close to the maximum speedup obtained by policy *All-4*. Policy *D4* gets a slightly lower value than policy *C2D4*. This means that there are cases in which C individuals execution time become the slowest ones executed with one core and this fact determines the execution time. Thus, in such cases, allocating C individuals to 2 cores would be suitable. However, if the efficiency (speedup/number of cores) is considered, *All-1* strategy presents a very poor efficiency since it needs 100 cores to be executed. So, it results in a very poor resource utilization. Figure 4(b) shows the average efficiency for each policy. Between the extreme policies *All-1* and *All-4*, executions of policy *D4* are using significantly less cores than policy *C2D4* achieving a minimally less speedup. It can be observed that assigning strictly 4 cores to D individuals (policy *D4*) provides better efficiency than policy *C2D4* where C individuals are assigned 2 cores. Figure 5(a) depicts the obtained speed up values for each case. As can be seen in Fig. 5(a), policies *D4* and *C2D4* almost reach the same speed up as the one obtained when allocating every simulation to 4 cores. In some cases, such as 1, 3 and 5, shortening class D individuals may cause that class C individuals become the lengthiest simulation at each generation. This

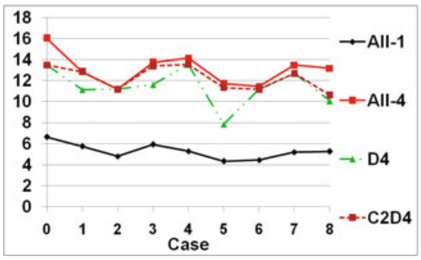


(a) Average Speedup.

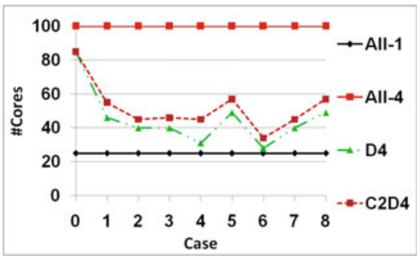


(b) Average Efficiency.

Fig. 4. Speedup and Efficiency for the 4 scheduling policies.



(a) Speedup per case



(b) Core usage per case

Fig. 5. Speedup and core usage for the 4 allocation policies.

can be observed in Fig. 5(b). The benefit obtained by the application of these simple policies can also be analyzed. For instance, considering cases 4 and 6, we are able to obtain almost the same speedup as the most favorable case, in terms of execution time (policy *All-4*) by adding only 3 and 6 cores, respectively. In general terms, policy *C2D4* requires only 7 cores extra to be implemented, but the gain is relevant.

5 Conclusions and Future Work

Nowadays, the existing forest fire spread simulation tools provide us with more accurate results as well as the ability to use more complex simulation features. However, the predictions based on a single-simulation strategy still presents the serious disadvantage of not being able to effectively deal with the uncertainty related to the context of an environmental emergency management. In this paper, we describe two kind of uncertainties which may lead us to inaccurate predictions: uncertainty in the values of the input parameters of the simulator, and uncertainty in the time needed to deliver a fire spread prediction. In order to deal with the former, we describe a two-stage prediction framework based on the iterative calibration of the input parameters according to the actual evolution of the fire. The later issue is tackled by means of an Artificial Intelligence technique (specifically, Decision Trees) to classify in advance the different sets of input parameters according to the expected execution time they will produce. However, while these methods clearly benefits the consecution of our goal, the need arises to address the problem of efficient computational resources exploitation. The proposed calibration technique for the two-stage prediction framework, GA, presents outstanding results as regards the improvement in the quality of our predictions, but it is very computational demanding, since it needs the execution of multiple simulations at the same time. Taking advantage of the fact that we are able to estimate how long a simulation will last before its execution, we propose to rely on this technique to decide how to allocate the different simulations to the available resources. The results obtained in the reported experimental study demonstrate the great benefit we obtain, mainly in terms of absolute time savings in the prediction process, by means of the application of simple core-allocation policies. This allows us to set out the main question of our ongoing work: the implementation of an intelligent scheduling system for the optimization of the available resources in order to enhance the quality of our predictions as quick as possible. For instance, it would be an important advantage to be able to dynamically group the fastest simulations in subsets of computational resources, allocating the slowest ones to other dedicated subsets, according to the specific needs of each case. Ongoing work is also related to add to our methodologies the capability to automatically adapt to new computing resources appearance in real time. The results obtained open up these new challenges with good expectations and a guaranteed background.

Acknowledgements. This research has been supported by MICINN-Spain under contract TIN2011-28689-C02-01.

References

1. Rothermel, R.C.: A mathematical model for predicting fire spread in wildland fuels. Director (INT-115), 40 p (1972)
2. Finney, M.A.: FARSITE, Fire Area Simulator-model development and evaluation. Res. Pap. RMRS-RP-4, Ogden, UT: U.S. Department of Agriculture, Forest Service, Rocky Mountain Research Station (1998)
3. Abdalhaq, B., Cortés, A., Margalef, T., Luque, E.: Enhancing wildland fire prediction on cluster systems applying evolutionary optimization techniques. Future Gener. Comput. Syst. **21**(1), 61–67 (2005)
4. Denham, M., Wendt, K., Bianchini, G., Cortés, A., Margalef, T.: Dynamic data-driven genetic algorithm for forest fire spread prediction. J. Comput. Sci. **3**(5), 398–404 (2012). (Advanced Computing Solutions for Health Care and Medicine)
5. Denham, M., Wendt, K., Bianchini, G., Cortés, A., Margalef, T.: Dynamic data-driven genetic algorithm for forest fire spread prediction. J. Comput. Sci. **3**(5), 398–404 (2012)
6. Fürlinger, K., Gerndt, M.: ompP: a profiling tool for OpenMP. In: Mueller, M.S., Chapman, B.M., de Supinski, B.R., Malony, A.D., Voss, M. (eds.) IWOMP 2005 and IWOMP 2006. LNCS, vol. 4315, pp. 15–23. Springer, Heidelberg (2008)
7. Graham, S.L., Kessler, P.B., McKusick, M.K.: gprof: a call graph execution profiler. SIGPLAN Not. **39**(4), 49–57 (2004)
8. Artés, T., Cencerrado, A., Cortés, A., Margalef, T.: Relieving the effects of uncertainty in forest fire spread prediction by hybrid mpi-openmp parallel strategies. Procedia Comput. Sci. **18**(2013), 2278–2287 (2013). (International Conference on Computational Science)
9. Bossard, M., Feranec, J., Otahel, J.: CORINE land cover technical guide Addendum 2000. Technical report No 40, European Environment Agency, Kongens Nytorv 6, DK-1050 Copenhagen K, Denmark (2000)
10. Cencerrado, A., Rodríguez, R., Cortés, A., Margalef, T.: Urgency versus accuracy: dynamic driven application system natural hazard management. Int. J. Numer. Anal. Model. **9**, 432–448 (2012)
11. Holmes, G., Donkin, A., Witten, I.H.: Weka: a machine learning workbench. In: Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems 1994. pp. 357–361. IEEE (1994)
12. Cencerrado, A., Cortés, A., Margalef, T.: On the way of applying urgent computing solutions to forest fire propagation prediction. Procedia Comput. Sci. **9**, 1657–1666 (2012)

The 4th Workshop on Performance Evaluation of Parallel Applications on Large-Scale Systems

The Effect of Parallelization on a Tetrahedral Mesh Optimization Method

Domingo Benitez(✉), Eduardo Rodríguez, José M. Escobar, and Rafael Montenegro

SIANI Institute, University of Las Palmas de Gran Canaria, Las Palmas de Gran Canaria, Spain
{dbenitez,erodriguez,jmescobar,rmontenegro}@siani.es

Abstract. A parallel algorithm for simultaneous untangling and smoothing of tetrahedral meshes is proposed in this paper. This algorithm is derived from a sequential mesh optimization method. We provide a detailed analysis of its parallel scalability and efficiency, load balancing, and parallelism bottlenecks using six benchmark meshes. In addition, the influence of three previously-published graph coloring techniques on the performance of our parallel algorithm is evaluated. We demonstrate that the proposed algorithm is highly scalable when run on a shared-memory computer with up to 128 Itanium 2 processors. However, some parallel deterioration is observed. Here, we analyze its main causes using a theoretical performance model and experimental results.

Keywords: Mesh optimization · Parallel performance evaluation

1 Introduction

Engineering design and analysis of real systems are becoming increasingly complex. The *80/20 design/analysis ratio* seems to be a common industrial experience: design and analysis account for about 80 % and 20 % of time, respectively. In the design process, there are many methods that are time consuming. On average, mesh generation accounts for 20 % of overall time and may take as much CPU-time as field solver, which may need of many man-months [18]. In our Meccano method for tetrahedral mesh generation, the most time-consuming phase is devoted to mesh optimization [15]. On the other hand, mesh generation tools frequently produce meshes with inverted and/or poorly shaped elements. So, untangling and/or smoothing techniques are applied to improve the mesh quality before or during the numerical analysis. In all these problems, improving the speed of mesh optimization with parallelism helps users solve problems faster.

In [8] we propose a simultaneous untangling and smoothing algorithm for tetrahedral meshes, in contrast with other techniques that require two separate and consecutive steps, one for untangling and another for smoothing. For very large tangled meshes, the runtime of our sequential algorithm may be long.

The improvement of our technique using parallel computation is not trivial because two vertices of the same tetrahedron cannot be simultaneously optimized on different processors. In this paper, we propose a parallel algorithm for simultaneously untangling and smoothing tetrahedral meshes with a number of vertices far greater than the number of available processors. Additionally, the performance of this algorithm on a shared-memory computer is analyzed. We show that our algorithm is highly scalable and compute-bound. However, the OpenMP thread scheduling overhead significantly deteriorates the parallel performance.

Section 2 summarizes our sequential approach to mesh untangling and smoothing. Section 3 describes its parallelization. The methodology we used to evaluate the performance of this parallel algorithm is explained in Sect. 4. Its results are presented in Sect. 5. Finally, Sect. 6 provides the main conclusions.

2 Our Approach to Tetrahedral Mesh Optimization

Let us consider $\mathcal{M}$ to be a tetrahedral mesh. Usual techniques to improve the quality of a mesh without inverted tetrahedra are based upon local smoothing [6]. These techniques consist of finding the new position $\boldsymbol{x}_v$ that each inner mesh node v must hold, in such a way that they optimize an objective function. Such a function is based on a certain measurement of the quality of the local submesh $N_v \subset \mathcal{M}$ that is formed by the set of tetrahedra connected to the *free node* v. As it is a local optimization process, we cannot guarantee that the final mesh is globally optimal. Nevertheless, after repeating this process several times for all the nodes of the mesh $\mathcal{M}$, quite satisfactory results can be achieved.

The algebraic quality metrics proposed by Knupp [13] provide us an appropriate framework to define objective functions. In this paper, we use,

$$K(\boldsymbol{x}_v) = \sum_{i=1}^{n} ([\eta_i(\boldsymbol{x}_v)]^p)^{\frac{1}{p}} \tag{1}$$

being n the number of elements in N_v, p is usually chosen as 1 or 2, $\eta_i = 1/q_i$ is the distortion of the i-th tetrahedron of N_v, and q_i is the mean ratio quality measure of a tetrahedron given by $q = 3\sigma^{\frac{2}{3}}/|S|^2$, where $|S|$ is the Frobenius norm of matrix S associated to the affine map from the ideal element (usually an equilateral tetrahedron) to the physical one, and σ is the determinant of matrix S: $\sigma = det(S)$. Specifically, the weighted Jacobian matrix S is defined as $S = AW^{-1}$, being $A = (\boldsymbol{x}_1 - \boldsymbol{x}_0, \boldsymbol{x}_2 - \boldsymbol{x}_0, \boldsymbol{x}_3 - \boldsymbol{x}_0)$ the Jacobian matrix, and $\boldsymbol{x}_j, j = 0, \ldots, 3$ the coordinates of the vertices of the tetrahedron. The constant matrix W is derived from the ideal element. For more details, see [8].

Objective functions like (1) do not work properly when there are inverted elements ($\sigma < 0$). This is because they present singularities (barriers) when any tetrahedron of N_v changes the sign of its Jacobian matrix. In [8] we proposed a suitable modification of the objective function such that it is regular all over $\mathbb{R}^3$. It consists of substituting the term σ in the quality metrics by the positive and increasing function $h(\sigma) = \frac{1}{2}(\sigma + \sqrt{\sigma^2 + 4\delta^2})$. When a feasible region exists (subset of $\mathbb{R}^3$ where a free node v could be placed, being its N_v a valid submesh), the minima of the original and modified objective functions are very close and, when this region does not exist, the minimum of the modified objective function is located in such a way that it tends to untangle N_v. With this approach, we can use any standard and efficient unconstrained optimization method to find the minimum of the modified objective function [1]. In this way, our method allows simultaneous untangling and smoothing of tetrahedral meshes, in contrast with other techniques that require two separate and consecutive steps.

3 Parallel Algorithm for Mesh Untangling and Smoothing

Algorithm 1 shows our sequential method for simultaneous untangling and smoothing of tetrahedral meshes. It has the following inputs: $\mathcal{M}$ is a tetrahedral mesh, `IT` is a function that provides its total number of inverted tetrahedra, N_v is the set of tetrahedra connected to a free node v, and $\boldsymbol{x}_v$ is the initial position of v. The algorithm iterates over all the nodes and adjusts the spatial coordinates $\boldsymbol{x}_v$ of a free node v in each step; $\hat{\boldsymbol{x}}_v$ is its position after optimization, which is provided by the procedure `OptimizeNode`. Then, Q saves the lowest quality of a tetrahedron when the above-mentioned quality function (q) is used (it is 0 if any tetrahedron is tangled). The function called `quality` measures the increment in Q between successive iterations of mesh optimization. The mesh is optimized until it is completely untangled: `IT`$(\mathcal{M}) = 0$, and successive iterations increase the minimum mesh quality less than $\lambda = 5\,\%$: $\Delta Q < \lambda$. The algorithm also stops when the number of optimization iterations is larger than `maxIter`.

Algorithm 1. Sequential algorithm for the mesh optimization method

```
1: $Q \leftarrow 0$
2: $j \leftarrow 0$
3: while (IT($\mathcal{M}$) $> 0$ or $\Delta Q \geq \lambda$) and $j \leq$ maxIter do
4:     for each vertex $v \in \mathcal{M}$ do
5:         $\hat{\boldsymbol{x}}_v \leftarrow$ OptimizeNode($\boldsymbol{x}_v$,$N_v$)
6:     end for
7:     $\Delta Q \leftarrow$ quality($\mathcal{M}$)
8:     $j \leftarrow j + 1$
9: end while
```

Algorithm 2 is a parallel algorithm for our mesh optimization method. Its inputs $\mathcal{M}$, `IT`, N_v, $\boldsymbol{x}_v$, `OptimizeNode`, `quality`, λ, and `maxIter` have the same meanings as described for Algorithm 1. This algorithm has to prevent two adjacent nodes from being simultaneously optimized in parallel. On the contrary, new inverted mesh tetrahedra may be created [10]. Thus, when the sequential Algorithm 1 is parallelized, a computational dependency appears between adjacent vertices. This justifies the use of a graph coloring technique in our parallel algorithm to find mesh vertices that do not have computational dependency.

Algorithm 2. Parallel algorithm for the mesh optimization method

```
1: $I \leftarrow$ Coloring(G=(V,E))
2: $Q \leftarrow$ 0
3: $j \leftarrow$ 0
4: while (IT($\mathcal{M}$) $> 0$ or $\Delta Q \geq \lambda$) and $j \leq$ maxIter do
5:     for each independent set $I_i \in I$ do
6:         for all each vertex $v \in \mathcal{M}$ do in parallel
7:             $\hat{\boldsymbol{x}}_v \leftarrow$ OptimizeNode($\boldsymbol{x}_v$,$N_v$)
8:         end for
9:     end for
10:    $\Delta Q \leftarrow$ quality($\mathcal{M}$)
11:    $j \leftarrow j + 1$
12: end while
```

We implemented graph coloring with procedure `Coloring`, which is expressed as follows. Let us consider $G = (V, E)$ to be the graph associated to the tetrahedral mesh $\mathcal{M}$, where V is its set of vertices (without spatial information), and E is the set of their edges, then `Coloring` is a procedure to color $G = (V, E)$. An *independent set* or *color*, I_i, is a set of non-adjacent vertices: $v \in I_i \rightarrow v \notin adj(I_i, G = (V, E))$, being $adj(I_i, G = (V, E))$ the set of vertices that are adjacent to all vertex $z \in I_i$, $z \neq v$. In this way, the graph $G = (V, E)$ of a tetrahedral mesh $\mathcal{M}$ is partitioned in a disjoint sequence of colors, $I = \{I_1, I_2, \dots\}$.

Three different and previously published coloring methods called *C1*, *C2*, *C3* were implemented. *C1* is a sequential coloring method that has been used for mesh smoothing [10]. It requires the use of the asynchronous coloring heuristic proposed in [12]. This heuristic is based on Luby's Monte Carlo algorithm for determining the maximal independent set [14]. *C2* is a parallel version of *C1* for shared-memory computers that was proposed by Jones and Plassmann [12] for distributed-memory computers. *C3* is an parallel greedy coloring algorithm that was proposed by Catalyurek et al. [5]. Section 5 compares the impact of these graph coloring methods on the performance of our parallel optimization method.

The vertex set with the same color (I_i) is equipartitioned among the available processors. Each processor optimizes its assigned set of vertices in a sequential fashion. At each sequential step, `OptimizeNode` procedure is applied to a single vertex v with the method described in Sect. 2. The new vertex spatial position is available to other processors by writing to shared memory. Each subsequent parallel phase optimizes another color until all vertices are optimized. Finally, the exit criteria are the same as the ones used for the sequential algorithm.

Previous studies on parallel algorithms for mesh optimization include the work of Freitag et al. [10], which relies on a theoretical shared-memory model with results using distributed-memory computers. Another similar study was published by Shontz and Nistor [17], which provides performance results for mesh simplification algorithms on GPUs although they do not use graph coloring algorithms to find mesh vertices that do not have computational dependency.

Table 1. Description of the tangled tetrahedral meshes

NAME	m	T	IT	DEGREE	OBJECT
"m = 6358"	6358	26446	2215	26	Bunny
"m = 9176"	9176	35920	13706	26	Tube
"m = 11525"	11525	47824	1924	26	Bone
"m = 39617"	39617	168834	83417	26	Screwdriver
"m = 201530"	201530	840800	322255	26	Toroid
"m = 520128"	520128	2201104	1147390	26	HR toroid

Legends. NAME: name of the tetrahedral mesh. m: total number of mesh vertices. T: total number of mesh tetrahedra. IT: total number of inverted tetrahedra. DEGREE: maximum vertex degree. OBJECT: short description of the real object that is meshed.

4 Experimental Methodology

Our experiments were conducted on a HP Integrity Superdome node with 128 Itanium 2 cores, multithreading disabled, and 1TB NUMA shared memory. Algorithms 1 and 2 were applied on six tangled tetrahedral meshes (see Table 1). All these meshes were constructed with a tool that applies an strategy for adaptive mesh generation based on the *Meccano* method [15], using surface triangulations from different repositories as input data [16]. We also used the Intel C++ compiler 11.1 with `-O2` on a Linux system. The source code of the parallel version included OpenMP directives, which were disabled when the serial version was compiled [9]. Both versions were profiled with PAPI [4], which uses performance counter hardware [11]. All versions were run with processor and memory binding enabled, and the processors were not shared among other user or system level workloads. For each mesh, we run the parallel version using a given maximum number of threads between 1 and 128. Since our algorithms are CPU-bound, there is little sense in using more threads than available cores. We use the following performance metrics: wall-clock time, true speedup, parallel efficiency, and load balancing, which were averaged over more than 30 runs. Each run was divided into two phases. The first phase loops over all vertices repetitively and

Table 2. Best execution times for the complete parallel algorithm (Algorithm 2)

NAME	SRT	BPRT	BNC	BSU	BPE (%)	BCA	C	I	MMQ	AMQ
"m = 6358"	17.3	1.5	72	11.7	16.2	*C1*	29	25	0.13	0.66
"m = 9176"	37.3	1.2	88	31.9	36.3	*C3*	29	26	0.26	0.68
"m = 11525"	33.7	1.1	120	29.7	24.8	*C3*	10	38	0.11	0.65
"m = 39617"	87.4	1.6	128	54.9	42.9	*C1*	31	11	0.17	0.73
"m = 201530"	2505.4	81.3	128	30.8	24.1	*C2*	21	143	0.23	0.67
"m = 520128"	2259.7	41.9	120	54.0	45.0	*C3*	34	36	0.22	0.68

Legends. NAME: mesh name. SRT: serial runtime (s). BPRT: best parallel runtime (s); its number of cores: BNC, speedup: BSU, parallel efficiency: BPE, coloring algorithm: BCA, colors: C, untangling/smoothing iterations: I, minimum mesh quality: MMQ, and average mesh quality: AMQ.

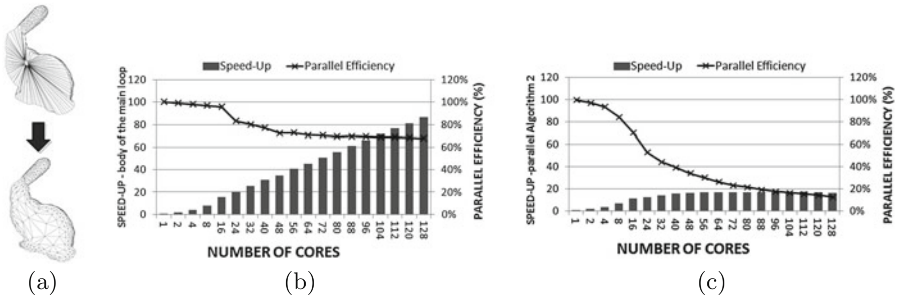


Fig. 1. (a) The tangled (upper) and untangled (lower) meshes called "m = 6358". True speedup and parallel efficiency of: (b) body of the main loop (line 7 of Algorithm 2), (c) complete parallel Algorithm 2, when the graph coloring technique is *C3*.

completely untangles a mesh; at the same time, the mesh is also smoothed. The second phase smoothes the mesh until the exit criteria is met. Table 2 shows the number of untangling and smoothing iterations (I) for each mesh.

5 Performance Evaluation

5.1 Performance Scalability

When the execution time of the main mesh optimization procedure is profiled in both versions of our algorithms, line 5 of sequential Algorithm 1 vs. line 7 of parallel Algorithm 2, we achieve results for true speedup as depicted in Fig. 1(b) when the "m = 6358" mesh is optimized. As can be seen, the true speedup linearly increases as the number of cores increases. Figure 1(b) also shows the parallel efficiency of the main mesh optimization procedure of Algorithm 2 (line 7). Note that up to 128 cores, the parallel efficiency is always above 67 %. Similar results were obtained for the rest of meshes. These results indicate that the main computation of our parallel algorithm is highly scalable. This performance scalability is caused by the parallel processing of independents sets of vertices (colors) that is made in each optimization iteration of the Algorithm 2 (line 7).

When the execution times of the complete sequential and parallel algorithms are profiled, we obtained results for true speedup as depicted in Fig. 1(c) when the "m = 6358" mesh is optimized. Note that in this case, the speedup does not increase so linearly as when the main mesh optimization procedures of algorithms are profiled, and maximum parallel efficiency is lower than 20 %. Table 2 shows the best results for all tetrahedral meshes. Note that the maximum parallel efficiency is 45 %, which was obtained when "m = 520128" mesh was optimized. Furthermore, the number of cores with best speedup depends on the benchmark mesh. In some cases, these highest performance results are obtained when the number of cores is lower than maximum (see column BNC in Table 2).

We investigated the causes of this degradation with a performance model. *SpeedUp* is modeled as the ratio between the sequential (T_S) and the parallel execution times when k cores are activated ($T_{P,k}$):

$$SpeedUp = \frac{T_S}{T_{P,k}} \tag{2}$$

being $T_{P,k}$ the sum of the idealized parallel execution time (T_k, without overhead and with perfect load balancing) and the average performance overhead (O_k) [3]:

$$T_{P,k} = T_k + O_k \quad ,, \quad T_k = \frac{T_S}{k} \quad \rightarrow \quad SpeedUp = \frac{k}{1 + \frac{O_k}{T_k}} \tag{3}$$

$$T_k = \frac{N_k}{IPC_k \times f} \ ,, \ O_k = \frac{N_{O,k}}{IPC_{O,k} \times f} \quad \rightarrow \quad SpeedUp = \frac{k}{1 + \frac{N_{O,k} \times IPC_k}{N_k \times IPC_{O,k}}} \tag{4}$$

N is the total number of executed instructions excluding no-operation instructions: N_k during mesh optimization tasks, and $N_{O,k}$ during parallel thread scheduling; IPC is the number of executed instructions per clock cycle during the runtime: IPC_k during mesh optimization tasks, and $IPC_{O,k}$ during parallel thread scheduling; and f is the fixed clock speed of 1.6 GHz. N and IPC are obtained during profiling with performance counter hardware [7,11].

In Fig. 2, real speedups (graphic marks) are overlapped with predictions of our performance model (lines) when three benchmark meshes are optimized. Real data were obtained using dynamic OpenMP thread scheduling and coloring algorithm *C3*. The average precision of the performance model for all meshes was 95.6 %. Based on these results, we can conclude that for our complete parallel Algorithm 2, true speedup and parallel efficiency deteriorate as the number of cores increases because they tend to be dominated by the loop-scheduling overhead. We note that this parallel overhead is caused by the Itanium 2 instructions ($N_{O,k}$) that are generated when the OpenMP directive that implements line 6 is compiled. Figure 2 also shows that the speedup is maximum at certain number of processors. Our model indicates that it is due to the quadratic polynomial form of the overhead time (O_k) as the number of threads increases.

5.2 Load Balancing

The error produced by the above-described model may be caused by other sources of performance degradation; for example, load imbalance between processors. The load imbalance of k cores (L_k) is measured as follows:

$$L_k = 100\,\% \times \frac{t_{max} - t_{min}}{t_{avg}} \quad ,, \quad t_{max} \geq t_{avg} \geq t_{min} > 0 \tag{5}$$

where t_{max}, t_{avg}, t_{min} are respectively the maximum, average and minimum execution times of the parallel threads without parallel loop-scheduling overhead (line 7 of Algorithm 2). As L_k is smaller, the difference between maximum and minimum thread execution time is smaller than the average execution time of

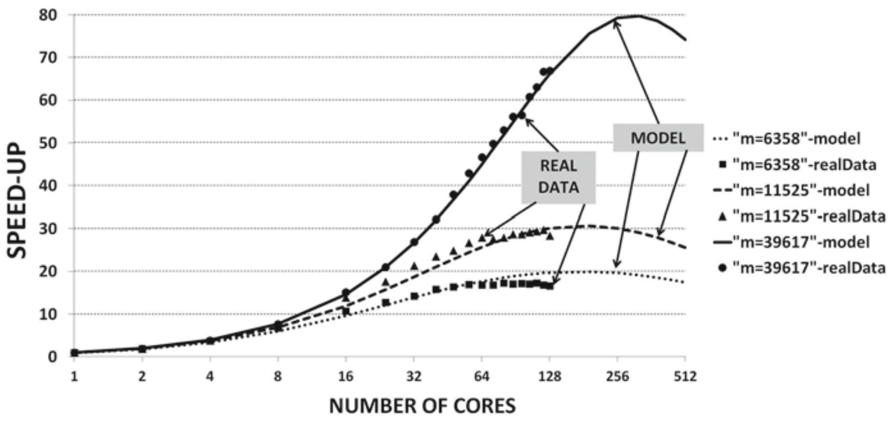


Fig. 2. Comparison of real data and data predicted by the performance model for the true speedup of parallel Algorithm 2.

threads. In these cases, threads tend to be stalled less time because load is more balanced, and so performance is better.

We note that the load imbalance of our parallel algorithm for the six benchmark meshes when all graph coloring algorithms and up to 128 Itanium 2 processors are used is mainly caused by the number of active threads. The higher the number of active threads, the higher the load imbalance. This means that as the loop-scheduling overhead instructions increase, the main computation of threads is more unbalanced. We tested our parallel algorithm on other x86 parallel computers, and the same effect on load imbalance was observed [2].

5.3 Parallelism Bottlenecks

We applied the profiling methodology described in [7] to analyze the performance inefficiencies caused by bottlenecks in processor functional units, cache memories, and NUMA shared memory. When up to 128 Itanium 2 processors are used, the stall cycles of parallel threads are in the range [29 %. . . 58 %]. These stall cycles are related to double-precision floating-point arithmetic units (from 70 % to 27 % of stall cycles), data loads (from 16 % to 55 %), and branch instructions (from 5 % to 14 %). Stall cycles due to data load instructions are mainly caused by cache memory latencies. NUMA memory latencies cause less than 1 % of data stall cycles, which is corroborated by monitoring the NUMA memory bandwidth usage that was never higher than 5 %. Thus, our parallel algorithm is compute bound and memory binding techniques have low impact on performance.

Another performance bottleneck was identified in the machine code that is generated by the compiler for Itanium 2 processors. On average, 40 % of executed instructions are no-operation instructions. This is caused by the long instruction format where up to three explicit instructions are included [11]. When the compiler cannot schedule a bundle of at least three instructions, no-operation instructions are used to fill some instruction slots. Enhancing the instruction level parallelism of our main optimization procedure may improve thread performance. However, it is a time-consuming task because the algorithms would have to be hand coded. Conventional x86 architectures do not suffer from this performance bottleneck because their instruction formats only include one instruction [2].

5.4 Influence of Graph Coloring Algorithms on Parallel Performance

Many papers evaluate the performance of graph coloring algorithms on parallel computers [5,10,12]. However, for the authors' knowledge, their impacts on the performance of algorithms that use these coloring algorithms is rarely reported.

First of all, we confirmed the performance results published in previous papers for *C1*, *C2* and *C3* coloring algorithms. Then, we investigated their influence on the performance of our parallel optimization algorithm. Since graph coloring aids in discovering parallelism, the time involved in graph coloring is only considered when profiling our parallel algorithm and not the sequential algorithm. When up to 128 processors and the six benchmark meshes are used, the percentage of total runtime that is required by the graph coloring methods *C1*, *C2* and *C3* ranges respectively from 0.8 % to 3.4 %, from 2.4 % to 12.9 %, and from 0.1 % to 1.9 %. This means that the computational load required by our parallel algorithm is much heavier than required by these graph coloring algorithms.

However, the total execution time depends on the selected coloring algorithm. We note that our parallel algorithm achieves the best performance on the Itanium2-based computer when the *C3* graph coloring technique is used. This is due to the lowest number of colors in which *C3* groups the vertices of each mesh with respect to the other graph coloring methods *C1* and *C2*. A lower number of vertex groups allow a larger number of vertices to be processed in parallel.

6 Conclusions and Future Work

We have proposed a new parallel algorithm for simultaneous untangling and smoothing of tetrahedral meshes. It is based on successive optimization iterations. In each of them, the spatial coordinates of independent sets of vertices are modified in parallel. The performance evaluation of this algorithm on a 128-core shared-memory computer using six meshes shows that it is a scalable and efficient parallel algorithm. It is due to the graph coloring method that is used to identify independent sets of vertices without computational dependency. We have additionally analyzed the causes of the parallel performance deterioration. We conclude that it is mainly due to loop-scheduling overhead of the OpenMP programming methodology. When analyzing hardware usage, we observe that our parallel algorithm is compute-bound because it uses the functional units and cache memory during 99 % of runtime. Finally, we also investigated the influence of three graph coloring methods on the performance of our parallel algorithm. They have low impact on the total execution time. However, the performance of our parallel algorithm depends on the selected coloring method. In this paper, we have shown that the *C3* coloring method [5] allows our parallel algorithm to achieve the highest parallel performance on a Itanium2-based computer.

The demonstrated scalability potential of our compute-bound parallel algorithm for shared-memory architectures encourages us to extend our work to achieve higher performance improvements from GPUs. The main problem will be to reduce the negative impact of global memory random accesses when non-consecutive mesh vertices are optimized by the same streaming multiprocessor.

Acknowledgments. This work has been supported by the Spanish *Sec. Estado Univ. e Inv.*, *Min. Economa y Competitividad* and *FEDER*, contract: CGL2011-29396-C03-01. It has been also supported by *Fondo Sec. CONACYT SENER Hidrocarburos*, contract: 163723, and two *CESGA ICTS* projects.

References

1. Bazaraa, M., Sherali, H., Shetty, C.M.: Nonlinear Programming. Wiley, New York (1993)
2. Benitez, D., Rodríguez, E., Escobar, J.M., Montenegro, R.: Performance evaluation of a parallel algorithm for simultaneous untangling and smoothing of tetrahedral meshes. In: Proceedings of the 22nd International Meshing Roundtable, pp. 579–598. Springer (2014)
3. Bronevetsky, G., Gyllenbaal, J., De Supinski, B.R.: CLOMP: accurately characterizing OpenMP application overheads. Int. J. Parallel Prog. **37**(3), 250–265 (2009)
4. Browne, S., Dongarra J., Garner N., London, K., Mucci, P.: A scalable cross-platform infrastructure for application performance tuning using hardware counters. In: Proceedings of the ACM/IEEE Conference on Supercomputing. IEEE Computer Society (2000)
5. Catalyurek, U.V., Feo, J., Gebremedhin, A.H., Halappanavar, M., Pothen, A.: Graph coloring algorithms for multicore and massively multithreaded architectures. Parallel Comput. **38**(10–11), 576–594 (2012)
6. Dompierre, J., Labbé, P., Guibault, F., Camarero, R.: Proposal of benchmarks for 3D unstructured tetrahedral mesh optimization. In: Proceedings of the 7th International Meshing Roundtable, pp. 459–478. Sandia National Laboratories (1998)
7. Ekman, P.: Studying program performance on the Itanium 2 with pfmon. www.pdc.kth.se/~pek/ia64-profiling.txt (2003)
8. Escobar, J.M., Rodríguez, E., Montenegro, R., Montero, G., González-Yuste, J.M.: Simultaneous untangling and smoothing of tetrahedral meshes. Comput. Methods Appl. Mech. Eng. **192**, 2775–2787 (2003)
9. Escobar, J.M., Cascón, J.M., Rodríguez, E., Montenegro, R.: A new approach to solid modeling with trivariate T-splines based on mesh optimization. Comput. Methods Appl. Mech. Eng. **200**(45–46), 3210–3222 (2011)
10. Freitag, L., Jones, M.T., Plassmann, P.E.: A parallel algorithm for mesh smoothing. SIAM J. Sci. Comput. **20**(6), 2023–2040 (1999)
11. Intel: Intel Itanium 2 processor reference manual (251110-003). Intel (2004)

12. Jones, M.T., Plassmann, P.E.: A parallel graph coloring heuristic. SIAM J. Sci. Comput. **14**(3), 654–669 (1993)
13. Knupp, P.M.: Algebraic mesh quality metrics. SIAM J. Sci. Comput. **23**(1), 193–218 (2001)
14. Luby, M.: A simple parallel algorithm for the maximal independent set problem. SIAM J. Comput. **4**, 1036–1053 (1986)
15. Montenegro, R., Cascón, J.M., Escobar, J.M., Rodríguez, E., Montero, G.: An automatic strategy for adaptive tetrahedral mesh generation. Appl. Numer. Math. **59**(9), 2203–2217 (2009)
16. Shape repositories. www.cyberware.com, http://graphics.stanford.edu/data/3Dscanrep, www-roc.inria.fr/gamma/gamma/download/download.php
17. Shontz, S.M., Nistor, D.M.: CPU-GPU algorithms for triangular surface mesh simplification. In: Proceedings of the 21st International Meshing Roundtable, pp. 475–492. Springer (2013)
18. Von Cottrell, J.A., Hughes, T.J.R., Bazilevs, Y.: Isogeometric Analysis: Toward Integration of CAD and FEA. Wiley, New York (2009)

Analysis of Partitioning Models and Metrics in Parallel Sparse Matrix-Vector Multiplication

Kamer Kaya[1], Bora Uçar[2](✉), and Ümit V. Çatalyürek[1,3]

[1] Department of Biomedical Informatics, The Ohio State University, Columbus, USA
{kamer,umit}@bmi.osu.edu

[2] CNRS and LIP, ENS Lyon, Lyon, France
bora.ucar@ens-lyon.fr

[3] Department of Electrical and Computer Engineering, The Ohio State University, Columbus, USA

Abstract. Graph/hypergraph partitioning models and methods have been successfully used to minimize the communication among processors in several parallel computing applications. Parallel sparse matrix-vector multiplication (SpMxV) is one of the representative applications that renders these models and methods indispensable in many scientific computing contexts. We investigate the interplay of the partitioning metrics and execution times of SpMxV implementations in three libraries: Trilinos, PETSc, and an in-house one. We carry out experiments with up to 512 processors and investigate the results with regression analysis. Our experiments show that the partitioning metrics influence the performance greatly in a distributed memory setting. The regression analyses demonstrate which metric is the most influential for the execution time of the libraries.

Keywords: Parallel sparse-matrix vector multiplication · Hypergraph partitioning

1 Introduction

Repeated sparse matrix-vector (SpMxV) and sparse matrix-transpose-vector multiplies that involve the same large, sparse matrix are the kernel operations in various iterative algorithms involving sparse linear systems. Such iterative algorithms include solvers for linear systems, eigenvalues, and linear programs. Efficient parallelization of SpMxV operations is therefore very important in virtually all large scale scientific computing applications. A number of partitioning methods and models based on hypergraphs have been used to enable efficient parallelization of SpMxV. These partitioning methods address different communication cost metrics for some variants of parallel SpMxV operations. In general, the importance of the communication cost metrics, such as the total volume of communication, the total number of messages and these two quantities on per processor basis, depends on the machine architecture, problem size, and the underlying parallel algorithm. In this study, we investigate the effects of the partitioning methods in order to identify the most relevant metrics and quantify their effects in various configurations. Our aims are to help the partitioner developers identify the important metrics, and to help the users of those partitioners to identify the most suitable partitioning method for their use case.

The standard hypergraph based models minimize the total volume of communication explicitly [7]. Some more recent variants do that while imposing a limit on the total number of communication by a 2D partitioning approach [6,9]. More sophisticated approaches [3,12,13] minimize different communication cost metrics on top of the total volume. Experimental investigations in these studies demonstrate that different communication cost metrics and their interplay can be important to achieve scalable parallel algorithms. It is therefore important to understand the effects of different metrics (optimized by different partitioning models) on the running time of applications under different configurations.

The contribution of this paper is two-fold. We designed and conducted several experiments in a system with 512 processors to show the effects of partitioning models and metrics on SpMxV performance. As far as we know, this is the first work which compares the existing partitioning models and metrics in modern architectures with modern software following message-passing paradigm. Our experiments confirm that it is difficult, if not impossible, to define the correct partitioning model and metric without analyzing the characteristics of the input matrices and the SpMxV library being used. We experimented with three existing libraries, PETSc [1,2], Trilinos [10], and an in-house library SpMV [14]. In order to overcome the mentioned difficulty, we carefully analyze the results using regression analysis techniques and relate the execution time of SpMxV implementations to different partitioning metrics. We portray this analysis, which forms out the second contribution, in detail so as to suggest improved objective functions for partitioning software and a guideline to choose partitioning methods for practitioners. Although, we only had an access to a 512-processor machine, the experiments and their analysis show that to scale larger systems, one needs to be more careful while partitioning the matrix—in our experiments the fact that the communication metrics greatly related to the execution time is observable starting from 64 processors.

2 Parallel SpMxV Operation and Software

Consider the sparse matrix-vector multiply operation of the form $\mathbf{y} \leftarrow \mathbf{A}\mathbf{x}$, where the nonzeros of the $m \times n$ matrix $\mathbf{A}$ are partitioned among K processors such that each processor P_k owns a mutually disjoint subset of nonzeros, $\mathbf{A}^{(k)}$ where $\mathbf{A} = \sum_k \mathbf{A}^{(k)}$. The vectors $\mathbf{y}$ and $\mathbf{x}$ are also partitioned among processors, where the processor P_k holds $\mathbf{x}^{(k)}$, a dense vector of size n_k, and it is responsible for computing $\mathbf{y}^{(k)}$, a dense vector of size m_k.

The standard parallel SpMxV algorithm [9,14,15] based on the described onzero and vector entry partitioning is called the *row-column-parallel* algorithm. In this algorithm, each processor P_k executes the following steps:

1. **Expand:** Send entries of $\mathbf{x}^{(k)}$ that are needed by others. Receive entries of $\mathbf{x}$ that are needed but owned by others.
2. **Scalar multiply-adds:** Perform $\bar{\mathbf{y}} \leftarrow \mathbf{A}^{(k)}\bar{\mathbf{x}}$, where $\bar{\mathbf{x}}$ contains $\mathbf{x}^{(k)}$ and the received entries of $\mathbf{x}$.
3. **Fold:** Send partial results from $\bar{\mathbf{y}}$ to the responsible processors. Receive contributions to the $\mathbf{y}^{(k)}$ vector.

If $\mathbf{A}$ is distributed columnwise, and the $\mathbf{x}$-vector entries are partitioned conformably with the column partition of $\mathbf{A}$, then the expand operation is not needed. Similarly, if $\mathbf{A}$ is distributed rowwise, and the $\mathbf{y}$-vector entries are partitioned conformably with the rows of $\mathbf{A}$, then the fold operation is not needed.

2.1 Libraries

There are different implementations of the above algorithm. We summarize three implementations with which we have experimented. Two of the implementations are in the well-known general libraries Trilinos [10] and PETSc [1,2]; the third one, SpMV, is an in-house library [14].

Algorithm 1. ParSpMxV-Trilinos variant

```
Input: $\mathbf{A}, x, \mu$
Output: $y$
1 SEND and RECEIVE $x$ vector entries so that each processor has the required
  $x$-vector entries
2 Compute $y_i^{(k)} \leftarrow a_{ij}\, x_j$ for the local nonzeros, i.e., the nonzeros for which
  $\mu(a_{ij}) = P_k$
3 SEND and RECEIVE local nonzero partial results $y_i^{(k)}$ to the processor $\mu(y_i) \neq P_k$,
  for all nonzero $y_i^{(k)}$
4 Compute $y_i \leftarrow \sum y_i^{\ell}$ for each $y_i$ with $\mu(y_i) = P_k$
```

Trilinos provides an implementation which can be described as in Algorithm 1 from the point of view of the processor P_k. In this implementation, the expand operations are finished before doing any computation. Then, all the scalar multiply-add operations are performed. Later on, the fold operations are completed. Trilinos uses `Irecv`/`Isend` and `waitall` communication primitives to handle the communications at steps 1 and 2 of Algorithm 1. It issues `Irecv`s, performs `Isend`s and then before commencing the computations ensures that all in the incoming data is received by using the `waitall` operation.

PETSc provides an implementation of the above algorithm only for the row-parallel case. Algorithm 2 summarizes that implementation from the point of view of P_k. First, the expand operation is initiated using `Irecv` and `Isend` primitives. Then, instead of waiting the reception of all necessary $\mathbf{x}$-vector entries, it performs some local computations so as to overlap communication and computations. In particular, the processor P_k performs scalar multiply-add operations

Algorithm 2. ParSpMxV-Overlap-PETSc variant

```
Input: $\mathbf{A}, x, \mu$
Output: $y$
1 SEND local $x_j$ (i.e., $\mu(x_j) = P_k$) to those processors that have at least one
  nonzero in column $j$
2 Compute $y_i^k \leftarrow a_{ij}\, x_j$ for the local nonzeros and local $x_j$, i.e., the nonzeros for
  which $\mu(a_{ij}) = P_k$ and $\mu(x_j) = P_k$
3 RECEIVE all non-local $x_j$ (i.e., $\mu(x_j) \neq P_k$)
4 Compute $y_i^k \leftarrow y_i^k + a_{ij}\, x_j$ for the local nonzeros and non-local $x_j$, i.e., the
  nonzeros for which $\mu(a_{ij}) = P_k$ and $\mu(x_j) \neq P_k$
```

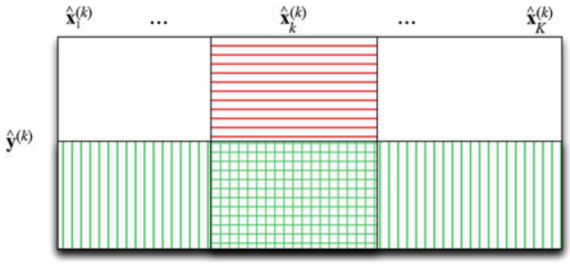

$\hat{\mathbf{x}}_1^{(k)}$
...
$\hat{\mathbf{x}}_k^{(k)}$
...
$\hat{\mathbf{x}}_K^{(k)}$
$\hat{\mathbf{y}}^{(k)}$

Fig. 1. The zones of the matrix $\mathbf{A}^{(k)}$ of processor P_k with respect to the vector $\mathbf{x}$ assuming a row-parallel algorithm.

using local a_{ij}'s for which $\mu(x_j) = P_k$ and there is no $a_{i\ell}$ with $\mu(x_\ell) \neq P_k$. Then, upon verifying the reception of all needed $\mathbf{x}$-vector entries using `waitall`, P_k continues with scalar multiply-add operations with the nonzeros on the rows that has at least one nonzero in a column j for which $\mu(x_\ell) \neq P_k$. The implementation can also be seen in an earlier technical report [11]. Figure 1 describes the algorithm pictorially. After issuing `Isends` and `Irecvs` (for $\hat{x}_\ell^{(k)}$), processor P_k performs the computations associated with the horizontally shaded matrix zone. Then, `waitall` is executed to have all $x^{(k)}$ before continuing with the rows that are below the horizontal ones. Note that the local matrices are actually permuted into the displayed form (local rows and the interface rows). The advantage of this implementation with respect to the Algorithm 1 is that it allows overlap between the reception of messages for the expand operation and scalar multiply-add operations with the nonzeros in local rows.

Consider again the matrix $\mathbf{A}^{(k)}$ of processor P_k as shown in Fig. 1. Before executing the `waitall` operation, there are some more scalar multiply-add operations that P_k can perform before the reception of any $\hat{x}_\ell^{(k)}$. These operations are related to the nonzeros that are in the hatched zone in the figure. In order to exploit the hatched zone for communication computation overlap, one can store that zone in the compressed column storage (CCS) format. This way, one can delay the invocation of the `waitall` operation for some more time. In fact, we can get rid of the `waitall` operation and maximize the communication computation overlap by performing all scalar multiply-operations that involve a received **x**-vector entry before waiting the reception of any other message. This requires storing the vertically shaded zones of the matrix in Fig. 1 in CCS format: with this, when P_k receives $\hat{x}_\ell^{(k)}$, it can visit the respective column and perform all operations. This way of storing the vertically shaded and hatched zones in CCS maximizes the amount of overlap in the strict sense (optimal amount of overlap) when a processor receives a single message from each sender (as should be the case in a proper SpMxV code). The third library that we investigate in this work, SpMV [14], implements this approach for row-parallel and row-column parallel algorithms (see descriptions in the accompanying technical report [8]).

2.2 Investigated Partitioning Metrics and Methods

We study the practical effects of five different metrics: the maximum number of nonzeros assigned to a processor (MaxNnz) which defines the load balance; the total communication volume (TotVol); the maximum send volume of a processor (MaxSV); the total number of messages (TotMsg); and the maximum number of messages sent by a processor (MaxMS). Our investigations are necessarily experimental, yet some a priori results can be told about these metrics, see the accompanying technical report [8] and the references therein.

We investigated the rowwise and columnwise partitioning methods CN and RN (the naming convention corresponds to the hypergraph models in the original paper [7]) among the one-dimensional partitioning approaches. Among the two-dimensional ones we investigated the fine grain (FG), and checkerboard (CB) partitioning models (see [9] and the references therein). These four methods try to reduce the total volume of communication and obtain load balance; the 2D methods implicitly reduce the total and maximum number of messages per processor. We also used a block partitioning (BL) model whose aim is just to have load balance in rowwise partitioning of the matrices. In this model, we traverse the rows from 1 to m, generate a part with approximately τ/K nonzeros, and continue with the next part when this number is exceeded. For matrices based on 2D meshes, we used another rowwise partitioning model called MP. This model tiles the 2D plane with a diamond-like shape [4, Sect. 4.8] and associates each shape (corresponding to a set of rows) with a processor. This approach balances the volume and the number of messages the processors send and receive.

3 Experimental Investigations

We carried our experiments on a 64-node cluster where each node has a 2.27 GHz dual quad-core Intel Xeon (Bloomfield) CPU and 48 GB main memory. Each core in a socket has 64 KB L1 and 256 KB L2 caches, and each socket has an 8 MB L3 cache shared by 4 cores. The interconnection network is 20 Gbps DDR InfiniBand. For parallelism, mvapich2 version 1.6 is used. We built SpMV, PETSc, and Trilinos with gcc 4.4.4 and used optimization flag -O3. For PETSc experiments, we used the matrix type `MPIAIJ` and the multiplication routine `MatMult`.

Table 1. Properties of the experiment matrices.

Matrix	Description	n	τ	Matrix	Description	n	τ
atmosmodl	Atmosp. model.	1,489,752	10,319,760	cage15	DNA electrop.	5,154,859	99,199,551
TSOPF_RS	Opt. pow. flow	38,120	16,171,169	HV15R	3D engine fan	2,017,169	283,073,458
Freescale1	Semicon. sim.	3,428,755	17,052,626	mesh-1024	5-point stencil	1,048,576	5,238,784
rajat31	Circuit sim.	4,690,002	20,316,253	mesh-2048	5-point stencil	4,194,304	20,963,328
RM07R	Comp. fluid dyn.	381,689	37,464,962	mesh-4096	5-point stencil	16,777,216	83,869,696

We used PaToH [5] with default setting `quality` for partitioning the matrices (this allows 0.03 imbalance). Since each node has 8 cores, we have 512 processors in total. In the experiments, we use $K \in \{1, 8, 16, 32, 64, 128, 256, 512\}$. For an experiment with $K \neq 1$ processors, we fully utilize $K/8$ nodes of the cluster. To measure the time of one SpMxV operation (in secs), we do 500 multiplications for each execution. The tables and figures show the averages of these 500 runs.

We used seven large real-life square matrices from different application domains that are available at the University of Florida (UFL) Sparse Matrix Collection (http://www.cise.ufl.edu/research/sparse/matrices) and three synthetically generated matrices corresponding to 5-point stencil meshes in 2D with sizes 1024×1024, 2048×2048, and 4096×4096. The properties of the matrices are given in Table 1.

In the experiments on real-life matrices, we use PETSc with rowwise models CN and BL, and SpMV and Trilinos with all models except MP. For meshes, we added MP to each library's model set. The reason is technological: PETSc provides SpMxV routines only for rowwise partitioning.

We designed a set of experiments to show the effect of different partitioning metrics in the actual running time of SpMxV computations. Our first two sets of experiments (Fig. 2 and Table 2 of the accompanying technical report [8]) showed clearly that the total volume of communication, the total number of messages, the maximum number and volume of messages sent by a processor affect significantly the running time of SpMxV. Which one of these four communication metric is the most important in different libraries? To what extent? What about their combinations? In order to answer these questions we carried out the following regression analysis.

3.1 Regression Analysis

To evaluate the performance of the libraries with respect to the partitioning metrics, we use linear regression analysis techniques and solve the nonnegative least squares problem (NNLS). In NNLS, given a variable matrix $\mathbf{V}$ and a vector $\mathbf{t}$, we want to find a dependency vector $\mathbf{d}$ which minimizes $\|\mathbf{Vd} - \mathbf{t}\|$ s.t. $\mathbf{d} \geq 0$. In our case, $\mathbf{V}$ has five columns which correspond to the partitioning metrics MaxNnz, TotVol, MaxSV, TotMsg, and MaxSM. Each row of $\mathbf{V}$ corresponds to

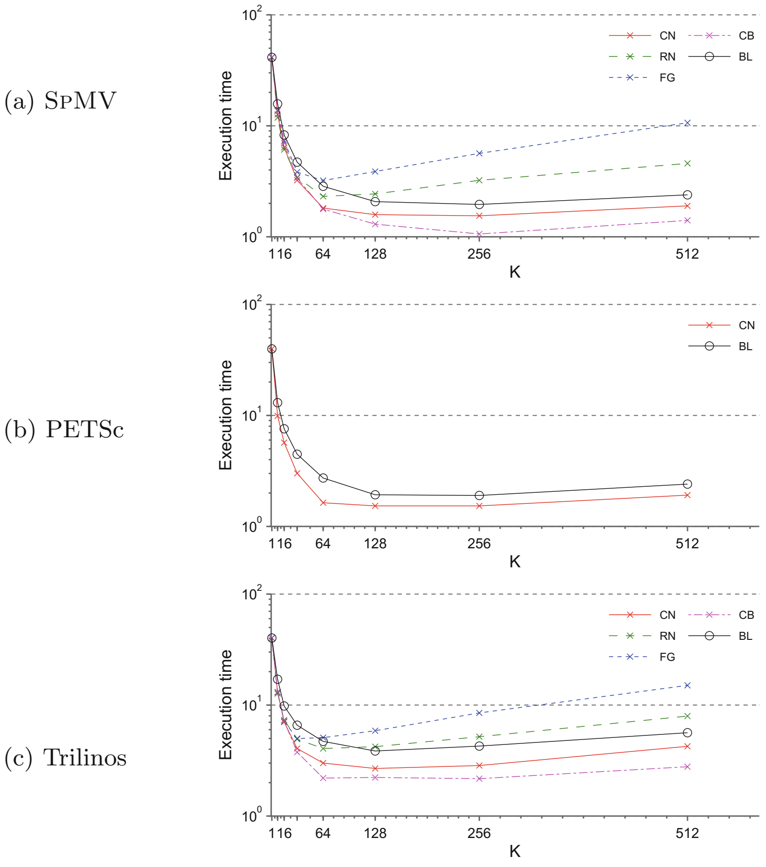


Fig. 2. Mean SpMxV times on real-life matrices in log scale for each library with respect to partitioning model.

an SpMxV execution where the execution time is put to the corresponding entry of **t**. Hence, we have the same **V** but a different **t** for each library. We apply a well-known technique in regression analysis and standardize each entry of **V** by subtracting its column's mean and dividing it to its column's standard deviation so that the mean and the standard deviation of each column become 0 and 1, respectively. This way, the units are removed and each column becomes equally important throughout the analysis. We then used MATLAB's `lsqnonneg` to solve NNLS. Each entry of the output d_i shows the dependency of the execution time to the partitioning metric corresponding to the ith column of **V**. Tables 2, 3, and 4 show the dependency values found in various settings.

We first apply regression analysis to each library with all matrices and row-wise partitioning models CN (column-net) and BL (block). The analysis shows that when $K \leq 64$, SpMxV performance depends rigorously on the maximum number of nonzeros assigned to a processor. In this case, the dependency values

Table 2. Regression analysis of SpMV, PETSc and Trilinos with all matrices and models CN and BL.

	$8 \le K \le 64$			$128 \le K \le 512$		
Metric	SpMV	PETSc	Trilinos	SpMV	PETSc	Trilinos
MaxNnz	8.02	7.81	6.80	0.49	0.44	0.83
TotVol	0.18	0.38	1.00	0.39	0.36	1.06
MaxSV	1.66	1.53	2.20	0.00	0.00	0.11
TotMsg	0.15	0.28	0.00	7.90	8.03	4.51
MaxSM	0.00	0.00	0.00	1.22	1.18	3.49

Table 3. Regression analysis of SpMV and Trilinos with all matrices and partitioning models. PETSc is not shown in this table because it cannot handle all the schemes.

	$8 \le K \le 32$		$64 \le K \le 128$		$256 \le K \le 512$	
Metric	SpMV	Trilinos	SpMV	Trilinos	SpMV	Trilinos
MaxNnz	8.43	7.54	2.75	2.52	0.00	0.02
TotVol	0.23	0.89	0.52	1.94	0.38	0.98
MaxSV	1.35	1.57	1.57	1.69	0.04	0.50
TotMsg	0.00	0.00	4.66	2.38	6.24	3.06
MaxSM	0.00	0.00	0.49	1.47	3.34	5.44

Table 4. Regression analysis of SpMV and Trilinos with mesh-based matrices and all partitioning models.

	$8 \le K \le 32$		$64 \le K \le 128$		$256 \le K \le 512$	
Metric	SpMV	Trilinos	SpMV	Trilinos	SpMV	Trilinos
MaxNnz	8.97	9.38	8.83	9.05	5.10	5.47
TotVol	0.00	0.00	0.00	0.24	0.00	0.00
MaxSV	0.72	0.48	0.43	0.09	0.92	0.52
TotMsg	0.00	0.00	0.42	0.07	0.42	0.99
MaxSM	0.31	0.14	0.33	0.55	3.55	3.02

for MaxNnz are 8.02, 7.81, and 6.80 for SpMV, PETSc, and Trilinos, respectively. As Table 2 shows, the next important metric is MaxSV with values 1.66, 1.53, and 2.20. The latency-based (TotMsg, MaxSM) partitioning metrics do not effect the performance for $K \le 64$. However, when K gets larger, these metrics are of utmost importance. Furthermore, the importance of MaxNnz decreases drastically for all the libraries. For SpMV and PETSc, MaxNnz becomes the 3rd important variable, whereas for Trilinos, it is the 4th. This shows that SpMV and PETSc handle the increase in the communication metrics better than Trilinos.

When $K \ge 128$, the dependency of Trilinos to TotMsg is much less than that of SpMV and PETSc. On the contrary, Trilinos' MaxSM dependency is almost 1.75 times more than SpMV and PETSc. This is expected since Trilinos uses Algorithm 1 which has no communication-computation overlap due to the use of `waitall` primitive. Such primitives can cause close coupling among the processors. When MaxNnz and the variance on the number of messages per processor are large, the overhead due to the bottleneck processor can result in poor SpMxV performance. Note that the dependency profiles of SpMV and PETSc, which are similar due to the communication-computation overlap, do not point out a similar bottleneck.

We extend the regression analysis to all matrices and all partitioning models and show the results in Table 3. The performance of SpMV and Trilinos rigorously depend on MaxNnz if $K \leq 32$, and on TotMsg and MaxSM when $K \geq 256$. Once again, Trilinos' MaxSM dependency is higher than that of SpMV due to the `waitall` primitive. To see the effect of matrix structure on regression analysis, we use only mesh-based matrices in the next experiment. As Table 4 shows, we observe that for these matrices, the performance of SpMV and Trilinos mostly depend on MaxNnz even when $K \geq 64$. Note that these matrices are banded and the communication metrics have relatively lower values compared to those of real-life matrices. Hence, the most dominant factor is MaxNnz.

In the light of the regression analysis experiments, we note that the partitioning metrics effect the performance of parallel SpMxV libraries. The best metric (or function of metrics) that needs to be minimized depends on the number of processors, the size and structure of the matrix, which we are planning to investigate in the future, and even the library itself. Although some of these variables are known while generating the partitions, predicting the others may need a preprocessing phase. For example, we already know that the libraries in this paper are employing point-to-point communication primitives which makes the connectivity metric suitable. However, if collective communication primitives, e.g., `MPI_ALLGATHER`, had been used, it would be better to minimize the cut-net metric as the main partitioning objective (however, we should note that such collective operations introduce unnecessary synchronization and messages especially for large K values). On the other hand, the matrix structure can be different for each input and a partitioner needs either a manual direction or a preprocessing to predict the best metric for each matrix.

3.2 Summary of Further Results

We provide summary of some further results and refer the interested reader to the accompanying technical report [8]. We observed that for all libraries and partitioning methods, minimizing TotMsg is more important than minimizing MaxSV for reducing the execution time, especially when K is large. Starting from $K = 64$, the difference becomes obvious in favor of TotMsg which is concordant with the regression analyses. For $K \in \{8, 16\}$, minimizing MaxSV or TotMsg are equally important.

The checkerboard method (CB) which is demonstrated (see Fig. 4 in [8]) to reduce most of the communication cost metrics seems to be the method of choice when K is not small (when K is small, there is no much difference between the models as also revealed by the regression analysis before). The relative performances of the partitioning methods do not change with the increasing K.

However, their difference tend to increase and hence, the model used for partitioning becomes more important as the parallel matrix-vector multiplication times of the libraries show in Fig. 2. When K is 256, the only significant reduction on the execution time is obtained by SpMV with the CB model.

4 Conclusion

We have carried out a detailed study to understand the importance of partitioning models and their effects in parallel SpMxV operations. As mentioned in the experiments, minimizing the right metric with the right partitioning model is crucial to increase throughput. For example, for the real-life matrices in our test set, CB model is the only one which can obtain a significant reduction on the SpMxV time when K is increased from 128 to 256 (after that we did not see any speed up). It is obvious that the other models fail to obtain such a reduction since the gain by dividing MaxNnz by two does not compensate the communication overhead induced by multiplying K by two. Hence, assuming the communication overhead is doubled on the average, doubling K increases the relative importance of communication on SpMxV four times.

Matrices from today's scientific and industrial applications can be huge. If one has only a few processors, partitioning may not matter, since the contribution of communication to the execution time will be low and the overall improvement on SpMxV via a good partitioning will be insignificant. However, as the regression analyses of Sect. 3.1 show, after a number of processors, the communication overhead will start to dominate the SpMxV time. For our experiments, this number is somewhere between 32 and 64, and it depends on the characteristics of the matrix, the library, and the architecture used for SpMxV operations. Although it may be more than 64, considering the advancements on CPU hardware, we can easily argue that this number will remain highly practical and partitioning will matter more for systems that are larger than those considered here.

References

1. Balay, S., Brown, J., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc users manual. Technical report ANL-95/11 - Revision 3.2, Argonne National Laboratory (2011)
2. Balay, S., Gropp, W.D., McInnes, L.C., Smith, B.F.: Efficient management of parallelism in object oriented numerical software libraries. In: Arge, E., Bruaset, A.M., Langtangen, H.P. (eds.) Modern Software Tools in Scientific Computing, pp. 163–202. Birkhäuser Press, Basel (1997)
3. Bisseling, R.H., Meesen, W.: Communication balancing in parallel sparse matrix-vector multiplication. Electron. Trans. Numer. Anal. **21**, 47–65 (2005)
4. Bisseling, R.H.: Parallel Scientific Computation: A Structured Approach using BSP and MPI. Oxford University Press, Oxford (2004)
5. Çatalyürek, Ü.V., Aykanat, C.: PaToH: A multilevel hypergraph partitioning tool, Version 3.0. Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH. http://bmi.osu.edu/umit/software.htm (1999)

6. Çatalyürek, Ü.V., Aykanat, C.: A hypergraph-partitioning approach for coarse-grain decomposition. In: Supercomputing'01 (2001)
7. Çatalyürek, Ü.V., Aykanat, C.: Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Trans. Parall. Distr. **10**(7), 673–693 (1999)
8. Çatalyürek, Ü.V., Kaya, K., Uçar, B.: On analysis of partitioning models and metrics in parallel sparse matrix-vector multiplication. Technical report INRIA, France (2013)
9. Çatalyürek, Ü.V., Aykanat, C., Uçar, B.: On two-dimensional sparse matrix partitioning: models, methods, and a recipe. SIAM J. Sci. Comput. **32**(2), 656–683 (2010)
10. Heroux, M.A., Bartlett, R.A., Howle, V.E., Hoekstra, R.J., Hu, J.J., Kolda, T.G., Lehoucq, R.B., Long, K.R., Pawlowski, R.P., Phipps, E.T., Salinger, A.G., Thornquist, H.K., Tuminaro, R.S., Willenbring, J.M., Williams, A., Stanley, K.S.: An overview of the trilinos project. ACM Trans. Math. Softw. **31**(3), 397–423 (2005)
11. Saad, Y., Malevsky, A.V.: P-SPARSLIB: A portable library of distributed memory sparse iterative solvers. Technical report umsi-95-180, Minnesota Supercomputer Institute, Minneapolis, MN (1995)
12. Uçar, B., Aykanat, C.: Minimizing communication cost in fine-grain partitioning of sparse matrices. In: Yazıcı, A., Şener, C. (eds.) ISCIS 2003. LNCS, vol. 2869, pp. 926–933. Springer, Heidelberg (2003)
13. Uçar, B., Aykanat, C.: Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. SIAM J. Sci. Comput. **25**(6), 1837–1859 (2004)
14. Uçar, B., Aykanat, C.: A library for parallel sparse matrix-vector multiplies. Technical report BU-CE-0506, Department of Computer Engineering, Bilkent University, Ankara, Turkey (2005)
15. Vastenhouw, B., Bisseling, R.H.: A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. SIAM Rev. **47**(1), 67–95 (2005)

Achieving Memory Scalability in the Gysela Code to Fit Exascale Constraints

Fabien Rozar[1,2(✉)], Guillaume Latu[1], and Jean Roman[3]

[1] IRFM, CEA Cadarache, 13108 Saint-Paul-les-Durance, France
fabien.rozar@cea.fr
[2] Maison de la simulation, CEA Saclay, 91191 Gif sur Yvette, France
[3] INRIA, Institut Polytechnique de Bordeaux, CNRS, 33405 Talence, France

Abstract. Gyrokinetic simulations lead to huge computational needs. Up to now, the semi-Lagrangian code Gysela performed large simulations using a few thousands cores (65k cores). But to understand more accurately the nature of the plasma turbulence, finer resolutions are wished which make Gysela a good candidate to exploit the computational power of future Exascale machines. Among the Exascale challenges, the less memory per core issue is one of the must critical. This paper deals with memory management in order to reduce the memory peak, and presents an approach to understand the memory behaviour of an application when dealing with very large meshes. This enables us to extrapolate the behaviour of Gysela for expected capabilities of Exascale machine.

Keywords: Exascale · Memory scalability · Memory footprint reduction · Plasma physics

1 Introduction

The architecture of the supercomputers will considerably change in the next decade. Since several years, CPU frequency does not increase anymore. Consequently the on-chip parallelism is dramatically increasing to offer more performance. Instead of doubling the clock-speed every 18–24 month, the number of cores per compute node follows the same law. These new parallel architectures are expected to exhibit different levels of memory and one tendency of these machines is to offer less and less memory per core. This fact has been identified as one of the Exascale challenges [11] and is one of our main concerns.

In the last decade, the simulation of turbulent fusion plasmas in Tokamak devices has involved a growing number of people coming from the applied mathematics and parallel computing fields [1]. These applications are good candidates to be part of the scientific applications that will be able to use the first generation of Exascale computers. The Gysela code already efficiently exploits supercomputing facilities [8]. In this paper we especially focus on its memory consumption. This is a critical point to simulate larger physical cases while using a constrained available memory.

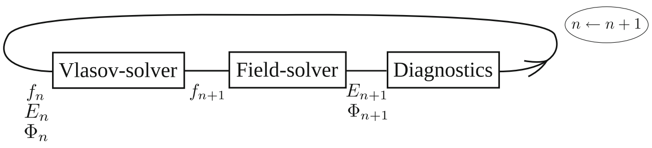


Fig. 1. Numerical scheme for one time step of Gysela

A module has been developed to provide a way to generate memory traces for the specific GYSELA application. However, our final goal (not achieved yet) is to define a methodology and a versatile and portable library to help the developer optimize memory usage in scientific parallel applications.

The goal of the work presented here is to decompose and reduce the memory footprint of GYSELA to improve its memory scalability. We present a tool which provides a visualization of the memory allocation/deallocation of GYSELA in off-line mode. An other tool allows us to predict the memory peak depending on some input parameters. This is helpful to check whether future simulation memory needs fit into available memory.

This article is organized as follow. Section 2 describes shortly the GYSELA code. Section 3 presents the memory consumption of GYSELA. Section 4 presents the module implemented to generate a trace file of allocation/deallocation in GYSELA. It also illustrates the visualization and prediction tool capabilities to handle the data of the trace file. Section 5 shows an example of reduction of the memory footprint and a study of the memory scalability thanks to the prediction tool. Section 6 concludes and presents some future works.

2 Overview of GYSELA

This section gives an overview of the global GYSELA algorithm and introduces the main data structures used.

GYSELA is a global nonlinear electrostatic code which solves a gyrokinetic Vlasov-Maxwell system. GYSELA is a coupling between a Vlasov solver that modelizes the motion of the ions inside a tokamak and a Maxwell solver that computes the electrostatic field which applies a force on the ions. The Vlasov equation is solved with a semi-Lagrangian method [6] and the Maxwell equation is reduced to the numerical solving of a Poisson-like equation [7].

In this gyrokinetic model, the main unknown is a distribution function f which represents the density of ions at a given phase space position. The execution of GYSELA is decomposed in the initialization phase, iterations over time, and the exit phase. Figure 1 illustrates the numerical scheme used during a time step. f_n represents the distribution function, Φ_n the electric potential and E_n the electric field which corresponds to the derivative of Φ_n. The Vlasov step performs the evolution of f_n over time and the Field-solver step computes E_n. Periodically, GYSELA executes diagnostics which export meaningful values extracted from f_n, E_n and saves the results in HDF5 files.

The distribution function f is a 5 dimensions variable and evolves over time. The first 3 dimensions are in space, $\mathbf{x}_G = (r, \theta, \varphi)$ with r and θ the polar coordinates in the poloidal cross-section of the torus, while φ refers to the toroidal angle. The two last coordinates are in velocity space: $v_\parallel$ the velocity along the magnetic field lines and μ the magnetic moment.

Let N_r, N_θ, N_φ, $\mathrm{N}_{v_\parallel}$ be respectively the number of points in each dimension r, θ, φ, $v_\parallel$. In the Vlasov solver, each value of μ is associated with a set of MPI processes (a MPI communicator). Within each set, a 2D domain decomposition allows us to attribute to each MPI process a sub-domain in (r, θ) dimensions. Thus, a MPI process is then responsible for the storage of the sub-domain defined by $f(r = [i_{start}, i_{end}], \theta = [j_{start}, j_{end}], \varphi = *, v_\parallel = *, \mu = \mu_{value})$. The parallel decomposition is initially set up knowing local values $i_{start}, i_{end}, j_{start}, j_{end}, \mu_{value}$. These 2D domains are derived from a classical block decomposition of the r domain into $\mathrm{p_r}$ sub-domains, and of the θ domain into p_θ sub-domains. The numbers of MPI processes used during one run is equal to $\mathrm{p_r} \times \mathrm{p}_\theta \times \mathrm{N}_\mu$. The OPENMP paradigm is then used in addition to MPI (#T threads in each MPI process) to use fine-grained parallelism.

3 Memory Bottleneck

3.1 Analysis

A GYSELA run needs a lot of amount of memory to be executed. During a run of GYSELA, each MPI process is associated with a μ value (Sect. 2) and sees the distribution function as a 4D array and the electric field as a 3D array. The remaining of the memory consumption is mostly related to arrays used to store precomputed values, MPI user buffers to concatenate data to send/receive and OPENMP user buffers to compute temporary results. Almost all the arrays are allocated during the initialization of GYSELA.

In order to better understand the memory behaviour of GYSELA, each allocation (allocate statement) is logged by storing: the array name, the type, and the size. Using these data we have done a *strong scaling* presented on the Table 1 (16 threads per MPI process). From the memory point of view, the *strong scaling* study consists in doing a run with a large enough mesh and evaluating the memory consumption for a process increasing step by step the number of MPI processes used for the simulation. If for a given simulation with n processes we use x Giga Bytes of memory per process, in the ideal case, one can hope that the same simulation with $2n$ processes would use $\frac{x}{2}$ Giga Bytes of memory per process. In this case, is it said that the memory *scalability* is perfect. But in practice, this is generally not the case because of parallelization memory overheads.

Table 1 shows the evolution of the memory consumption with the number of cores for a MPI process. The percentage of memory consumption compared with the total memory of the process is given for each type of data structures. The dimensions of the mesh are set to: $N_r = 1024$, $N_\theta = 4096$, $N_\varphi = 1024$, $N_{v_\parallel} = 128$, $N_\mu = 2$. This mesh is bigger than the meshes used in production

Table 1. Strong scaling: static allocation sizes in (GB per Mpi process) and percentage of the total for each kind of data

Number of cores	2k	4k	8k	16k	32k
Number of Mpi processes	128	256	512	1024	2048
4D structures	209.2	107.1	56.5	28.4	14.4
	67.1 %	59.6 %	49.5 %	34.2 %	21.3 %
3D structures	62.7	36.0	22.6	19.7	18.3
	20.1 %	20.0 %	19.8 %	23.7 %	27.1 %
2D structures	33.1	33.1	33.1	33.1	33.1
	10.6 %	18.4 %	28.9 %	39.9 %	49.0 %
1D structures	6.6	3.4	2.0	1.7	1.6
	2.1 %	1.9 %	1.7 %	2.0 %	2.3 %
Total per MPI process in GBytes	311.5	179.6	114.2	83.0	67.5

nowadays, but match further needs, especially those expected for multi-species physics. The last case with 2048 processes requires 67.5 GB of memory per Mpi process. We usually launch a single Mpi process per node. One can notice the memory required is much more than the 64 GB of a Helios[1] node or than the 16 GB of a Blue Gene/Q node. Table 1 also illustrates that 2D structures and many 1D structures do not benefit of the domain decomposition. In fact, the memory cost of the 2D structures does not depend on the number of processes at all, but rather on the mesh size and the number of threads. On the last case with 32k cores, the cost of the 2D structures is the main bottleneck. It takes 49 % of the whole memory footprint.

In Gysela, the memory overhead for large simulations is due to various reasons. Extra memory can be needed, for example to store some coefficients during an interpolation (for the Semi-Lagrangian solver of the Vlasov equation). Mpi buffers appear also as memory overhead. The Mpi subroutines accept as input 1D array which often requires to copy the data we want to send or receive in an appropriate way. We have reduced some of these memory overheads. It has improved the memory scalability and has allowed us to run bigger physical cases.

3.2 Approach

There are two ways to reduce the memory footprint of a parallel application. On the one hand one can increase the number of nodes used for the simulation. Since the size of structures which benefit of a domain decomposition will decrease along with the number of Mpi processes. On the other hand, we can manage more finely the allocations of arrays in order to reduce the memory costs that do not scale with the number of threads/Mpi processes and to limit the impact of all allocated data at the memory peak.

To achieve the reduction of the memory footprint and to push back the memory bottleneck, we choose to focus on the second approach.

[1] http://www.top500.org/system/177449

In the original version of the code, most of the variables are allocated during the initialization phase. This approach is rightful for structures which are *persistent variables* in opposition to *temporary variables* that could be dynamically allocated. In this configuration, firstly, one can determine early the memory space required without actually executing a complete simulation. This allows a user to know if the case submitted can be run or not. Secondly, it avoids execution overheads due to dynamic memory management. But a disadvantage of this approach is that variables used locally in one or two subroutines consume their memory space during the whole execution of the simulation. As the memory space becomes a critical point when a large number of cores are employed, we have allocated a large subset of these as temporary variables with dynamic allocation. This has reduced the memory peak with a negligible impact on the execution time. Also, we notice that some persistent variables can be deallocated at the memory peak time which can decrease memory footprint. However, one issue with dynamic allocations is that we lost the two main advantages of the static allocations, and particularly the ability to determine in advance the memory space required to run a simulation.

4 Customised Modeling and Tracing Memory Tools

To follow the memory consumption of Gysela and to measure the memory footprint reduction, three different tools has been developed: a Fortran module to generate a trace file of allocations/deallocations, and a visualization + prediction Python script which exploits the trace file. The information retrieved from the execution of Gysela thanks to the instrumentation module is a key component of our memory analysis. The implementation of these helpful tools is detailed in the following sections.

4.1 Trace File

Various data structures are used in Gysela, and in order to handle their allocations/deallocations, a dedicated Fortran module was developed to log them to a file: the *dynamic memory trace*. As the Mpi processes have almost the same dynamic memory trace, in the current implementation, we produce a single trace file for the allocations/deallocations of the Mpi process `0`.

Overview. In the community of performance analysis tools dedicated to parallel application, different approaches exist. But almost all of them relies on *trace files*. A trace file collects information from the application to represent one aspect of its execution: execution time, number of Mpi messages send, idle time, memory consumption and so on. But to obtain these information, the application have to be instrumented. The instrumentation can be made at 4 levels: in the source code, at the compilation time, at the linking step or during the execution (just in time).

The Scalasca performance tool [5] is able to instrument at the compilation time. This approach has the advantage to cover all code parts of the application and it allows the customization of the retrieved information. This systematic approach gives a full detailed trace but the record of information in all subroutine of the code may induce a consequent overhead in execution time. Also with an automatic instrumentation, it would be difficult to retrieve the expression of an allocation, like we do (cf. next section). The tool set EZTrace [2] offers the possibility to intercept calls to a set of functions. This tool can quickly instrument an application thanks to a link with third-party libraries at the linking step. Unlike our approach, this one does not need an instrumentation of the code but you cannot hope to retrieve the allocation expression in this approach. The tools Pin [9], DynamoRIO [3] or Maqao [4] produce an instrumentation during the execution time. The advantage here is the generic aspect of the method. Any program can be instrumented this way, but unlike our approach, these ones often introduce a quite large overhead of execution time.

The tool we have developed allows us to measure the performance of Gysela, from the memory point of view. A visualization tool has been developed to deal with the provided trace file. It offers a global view of the memory consumption and an accurate view around the memory peak to help the developer to reduce the memory footprint. The terminal output of the post processing script gives precious information about the arrays allocated at the memory peak. Given a trace file, we can also extrapolate the memory consumption in function of the input parameters. This allows us to investigate the memory scalability. As far as we know, there is no equivalent tool to profile the memory behaviour in the HPC community.

Implementation. A dedicated Fortran module of instrumentation has been developed. This instrumentation will generate a trace file. Then we practice a post-mortem analysis on it. The instrumentation module offers an interface, *take* and *drop*, which wraps the calls to `allocate` and `deallocate`. The `take` and `drop` subroutines perform the allocation and deallocation of the array handled and they log their memory action in the dynamic memory trace file.

For each allocation and deallocation, the module logs the name of the array, its type, its size and the expression of number of elements. The expression is required to make prediction. For example, the expression associated to this allocation:

```
integer, dimension(:,:), pointer  :: array
integer                           :: a0, a1, b0, b1
allocate(array(a0:a1, b0:b1))
```

is

$$(a1 - a0 + 1) \times (b1 - b0 + 1) \tag{1}$$

To be able to evaluate these allocation expressions, the variables inside them must be recorded. Either the value of the variable is logged, either an arithmetical

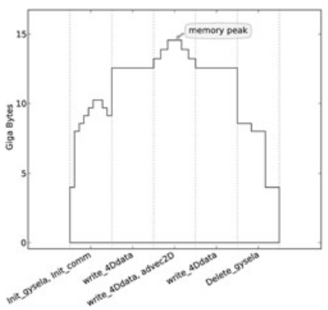


Fig. 2. Evolution of the dynamic memory consumption during GYSELA execution

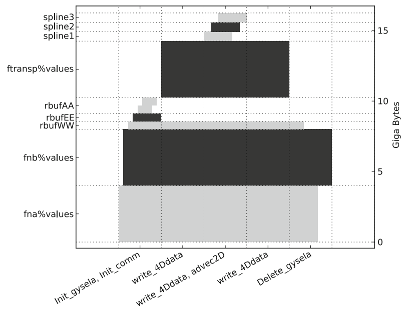


Fig. 3. Allocation and deallocation of arrays used in different GYSELA subroutines

expression depending on other recorded variables is logged. This is done respectively by the subroutines *write_param* and *write_expr*. The writing of expression saves the relationship between parameters in the trace file. This is essential for the prediction tool (Sect. 4.3). The following code is an example of recording the parameters $a0, a1, b0, b1$:

```
call write_param('a0', 1); call write_param('a1', 10)
call write_param('b0', 1); call write_expr('b1', '2*(a1-a0+1)')
```

To retrieve the temporal aspect of the memory allocation, the entry/exit to selected subroutines is recorded by the interface *write_begin_sub* and *write_end_sub*. This allows us to localize where happen the allocations/deallocations which is an essential aspect for the visualization step.

4.2 Visualization

In order to address memory consumption, we have to identify the parts of the code where the memory usage reaches its peak. The log file can be large, some Mega Bytes. To manage this amount of data, a PYTHON script was developed to visualize them. This tool will help the developer to understand the memory cost of the handled algorithms, and so give him some hints how and where it is meaningful to decrease the memory footprint. These information are given thanks to two kinds of plot.

Figure 2 plots the *dynamic* memory consumption in GB along time. The X axis represents the chronological entry/exit of instrumented subroutines. The Y axis gives memory consumption in GB. Figure 3 shows which array is used in which subroutine. The X axis remains identical as previously and the Y axis shows a name of array. Each array is associated to a horizontal line of the picture. The allocation of an array matches a rectangular filled in dark or light grey color in its corresponding line. The width of rectangles depends on the subroutines where allocation/deallocation happens.

In Fig. 2 one can locate in which subroutine the memory peak is reached. In Fig. 3 one can then identify the arrays that are actually allocated when the memory peak is reached. Thanks to these information, we exactly know where to modify the code in order to reduce the global memory consumption.

4.3 Prediction

To anticipate our memory requirements to run a given simulation, we need to predict the memory consumption for a given input parameter set. Thanks to the expressions of array size and the value or expression of numerical parameters contained in the trace file, we can model the memory behaviour off-line. The idea here is to reproduce allocations with any input set of parameters.

Sometimes, a parameter value cannot be expressed as a one line arithmetical expression (e.g. multi criteria optimization loop to determine the value). To manage this case and in order to be faithful, the Fortran piece of code which returns the value is call from Python script. This is possible thanks to a compilation of the Fortran needed sources with `f2py` [10].

By changing the value of input parameters, our prediction Python tool offers the possibility to extrapolate the Gysela memory consumption on greater meshes and even on supercomputer configurations which do not exist yet, as the Exascale ones. The results of this tool are presented in the Sect. 5.2.

5 Results

5.1 Memory Footprint Reduction

Reduce the memory footprint is equivalent to cut down the memory peak. The Figs. 4 and 5 show the impact on memory of some modifications of the code. After analysis of the code, we noticed that during the memory peak, the transposition structure `ftransp%values` and the distribution function `fnb%values` contain the same data organized differently. We obtain the trace of the Fig. 5 in deallocating `fnb%values` during the memory peak.

With this tool, one can see that depending on the size of the mesh and the number of Mpi processes and OpenMP threads, the memory peak moves. This behavior can be explained by the dependencies between the size of some characteristic arrays and the value of some input parameters. For example, Mpi buffer sizes are sensitive to parallelization parameters. In Gysela, the sizes of temporary buffer are sensitive to the number of points in r and *theta* dimensions.

The visualization tool gives a new point of view of the source code. This tool helped us to iteratively reduce the memory overhead and thus to improve the memory scalability.

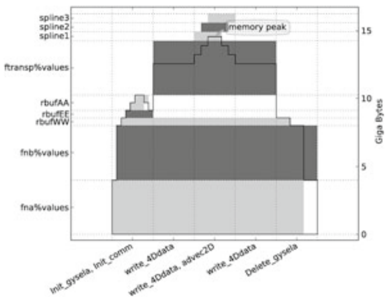


Fig. 4. First trace visualization

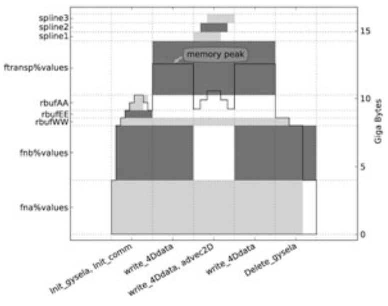


Fig. 5. Second trace visualization

5.2 Prediction over Large Meshes

Scalability. The Table 2 presents the strong scaling test with the new dynamic allocations, and several algorithmic improvements we have done thanks visualization tool (not detailed here). The prediction tool allows us to reproduce the Table 1 on the same mesh, i.e. $N_r = 1024$, $N_\theta = 4096$, $N_\varphi = 1024$, $N_{v_\parallel} = 128$, $N_\mu = 2$.

Table 2. Strong scaling: memory allocation size and percentage of the total for each kind of data at the memory peak moment

Number of cores	2k	4k	8k	16k	32k
Number of Mpi processes	128	256	512	1024	2048
4D structures	207.2	104.4	53.7	27.3	14.4
	79.2 %	71.5 %	65.6 %	52.2 %	42.0 %
3D structures	42.0	31.1	18.6	15.9	11.0
	16.1 %	21.3 %	22.7 %	30.4 %	32.1 %
2D structures	7.1	7.1	7.1	7.1	7.1
	2.7 %	4.9 %	8.7 %	13.6 %	20.8 %
1D structures	5.2	3.3	2.4	2.0	1.7
	2.0 %	2.3 %	3.0 %	3.8 %	5.1 %
Total per Mpi process in GBytes	261.5	145.9	81.9	52.3	34.3

The Table 2 outputs the memory consumption at the memory peak. It is obtained by keeping the mesh size constant and changing the number of Mpi processes and OpenMP threads. The prediction script replays the allocation/deallocation of trace file with the new parameters. As you can see on the bigger case (32k cores), the consumption of the 2D structures were reduced by 20.8 %. Also the memory gain on this case is of **50.8 %** on the global consumption relatively to Table 1. The 4D structures contain the most relevant data used during the computation, and they consume the major part of the memory as they should. The memory overheads have been globally reduced which improves the memory scalability of Gysela and allows larger simulations to be run.

Investigation. By using the prediction tool, larger meshes can be investigated and the size of the machine required to handled this amount of data can be estimated. With the actual implementation, to run the mesh $N_r = 2048$, $N_\theta = 4096$, $N_\varphi = 2048$, $N_{v_\parallel} = 256$, $N_\mu = 2$, the number of cores needed is of 524k cores, with 64 GB per process and 16 threads per process.

6 Conclusion

The work described in this paper focuses on a memory modeling and tracing module and some post processing tools which enable one to improve the memory scalability. With this framework, the understanding of the memory footprint behaviour along time is accessible. Also, the generated trace file can be reused to extrapolate the memory consumption for different input sets of parameter in off-line mode; this aspect is important both for end-user who needs greater resolutions or features with greedy memory needs, and for developer to design algorithms for Exascale machine.

With these tools, a reduction of **50.8 %** of the memory peak has been achieved and the memory scalability of the Gysela has been improved. Our next objective is to implement a versatile C/Fortran library. The work presented in this paper is a first step toward building a methodology that helps developers to improve memory scalability of parallel applications.

References

1. Åström, J.A., et al.: Preparing scientific application software for exascale computing. In: Manninen, P., Öster, P. (eds.) PARA 2012. LNCS, vol. 7782, pp. 27–42. Springer, Heidelberg (2013)
2. Aulagnon, C., Martin-Guillerez, D., Rué, F., Trahay, F.: Runtime function instrumentation with EZTrace. In: Caragiannis, I., et al. (eds.) Euro-Par Workshops 2012. LNCS, vol. 7640, pp. 395–403. Springer, Heidelberg (2013). http://link.springer.com/chapter/10.1007/978-3-642-36949-0_45
3. Bruening, D., Garnett, T., Amarasinghe, S.: An infrastructure for adaptive dynamic optimization. In: CGO 2003, pp. 265–275. IEEE (2003). http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1191551
4. Djoudi, L., Barthou, D., Carribault, P., Lemuet, C., Acquaviva, J.T., Jalby, W., et al.: Maqao: modular assembler quality analyzer and optimizer for itanium 2. In: The 4th Workshop on EPIC Architectures and Compiler Technology, San Jose (2005). http://www.labri.fr/perso/barthou/ps/maqao.pdf
5. Geimer, M., Wolf, F., Wylie, B.J., Ábrahám, E., Becker, D., Mohr, B.: The scalasca performance toolset architecture. CCPE **22**(6), 702–719 (2010). http://onlinelibrary.wiley.com/doi/10.1002/cpe.1556/full
6. Grandgirard, V., Sarazin, Y., Garbet, X., Dif-Pradalier, G., Ghendrih, P., Crouseilles, N., Latu, G., Sonnendrucker, E., Besse, N., Bertrand, P.: Computing ITG turbulence with a full-f semi-Lagrangian code. Commun. Nonlinear Sci. Numer. Simul. **13**(1), 81–87 (2008)
7. Hahm, T.S.: Nonlinear gyrokinetic equations for tokamak microturbulence. Phys. Fluids **31**(9), 2670–2673 (1988)

8. Latu, G., Grandgirard, V., Crouseilles, N., Dif-Pradalier, G.: Scalable quasineutral solver for gyrokinetic simulation. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011, Part II. LNCS, vol. 7204, pp. 221–231. Springer, Heidelberg (2012)
9. Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: building customized program analysis tools with dynamic instrumentation. In: ACM SIGPLAN Notices, vol. 40, pp. 190–200. ACM (2005). http://dl.acm.org/citation.cfm?id=1065034
10. Peterson, P.: F2py: a tool for connecting fortran and python programs. Int. J. Comput. Sci. Eng. **4**(4), 296–305 (2009). http://cens.ioc.ee/~pearu/papers/IJCSE4.4_Paper_8.pdf
11. Shalf, J., Dosanjh, S., Morrison, J.: Exascale computing technology challenges. In: Palma, J., Daydé, M., Marques, O., Lopes, J. (eds.) VECPAR 2010. LNCS, vol. 6449, pp. 1–25. Springer, Heidelberg (2011). https://www.nersc.gov/assets/NERSC-Staff-Publications/2010/ShalfVecpar2010.pdf

Probabilistic Analysis of Barrier Eliminating Method Applied to Load-Imbalanced Parallel Application

Naoki Yonezawa[1](✉), Ken'ichi Katou[1], Issei Kino[1], and Koichi Wada[2]

[1] Kanagawa University, Hiratsuka, Kanagawa, Japan
yonezawa@info.kanagawa-u.ac.jp
[2] University of Tsukuba, Tsukuba, Ibaraki, Japan

Abstract. In order to reduce the overhead of barrier synchronization, we have proposed an algorithm which eliminates barrier synchronizations and evaluated its validity experimentally in our previous study. As a result, we have found that the algorithm is more effective to the load-imbalanced program than load-balanced program. However, the degree of the load balance has not been discussed quantitatively. In this paper, we model the behavior of parallel programs. In our model, the execution time of a phase contained in a parallel program is represented as a random variable. To investigate how the degree of the load balance influences the performance of our algorithm, we varied the coefficient of variation of probability distribution which the random variable follows. Using the model, we evaluated the execution time of parallel programs and found that theoretical results are consistent with experimental ones.

Keywords: Barrier elimination · Probabilistic analysis · Time reduction

1 Introduction

Since barrier synchronization is a simple means to guarantee the order of data producing and data consuming, it is often used in parallel programs. However, barrier synchronization causes the processors' idle time to increase. To reduce the overhead of barrier synchronization, several methods [1–4] have been proposed. In our method [3,4] which targets on the distributed shared memory environment realized on a PC cluster, the compiler analyzes the dependency between the data producers and the data consumers. Then, the compiler replaces the barrier synchronization with message passing code which sends the data to consumer's side. Through evaluation, we have found that the algorithm is more effective to the load-imbalanced program than load-balanced program. However, the degree of the load balance has not been discussed quantitatively.

In this paper, in order to analyze the effect of the method theoretically, we propose probabilistic model to describe the degree of the load balance and evaluate the effectiveness of a barrier eliminating algorithm in terms of the load balance.

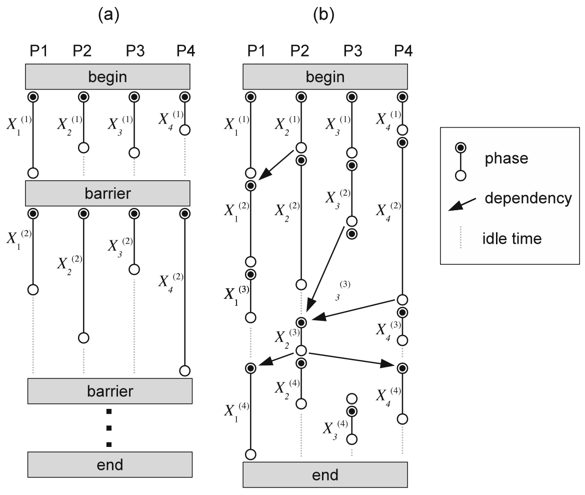


Fig. 1. Eliminating barriers

The rest of this paper is organized as follows: in Sect. 2, we propose a probabilistic model to investigate the barrier eliminating algorithm. With the model, one can describe data dependencies among processors as well as the degree of the load balance. We also show how to compute the execution time of a parallel program mathematically. Using the results obtained from our model, Sect. 3 discusses the effect of the barrier eliminating algorithm when applied to three typical dependency patterns. In Sect. 4, we describe related work. Finally, we conclude our study and describe our future work in Sect. 5.

2 A Probabilistic Analysis of a Barrier Eliminating Algorithm

In this section, at first, we define a behavioral model of a parallel program and introduce the mathematical symbols for discussion. Then, we construct the probabilistic model which describes the behavior of parallel program. Finally, we introduce the coefficient of variation to represent the degree of the load balance among processors.

2.1 The Behavioral Model of Parallel Program

Before Eliminating Barriers. Figure 1(a) shows the behavioral model of a parallel program on which this paper targets. In this model, a program contains a loop whose iteration has one and only barrier synchronization. We call an iteration of the loop a *phase*. The task in a phase is divided into n subtasks and the subtasks are assigned to n processors. At runtime, when a processor arrives a barrier synchronization, the processor stalls. After all the other processors arrives the barrier synchronization, all processors execute the next phases.

It is expected that the execution times of a phase are equal among processors if the phase contains the equally-divided subtasks and a barrier synchronization. However, the execution of a parallel program in the real world is influenced by the external factors including cache misses and the network delay. These cause *randomness*, i.e., the execution time of a processor in a phase can differ from the one of another processor in the same phase. In order to model the execution of such programs, we denote the execution time of Processor j in Phase i by $X_j^{(i)}$, $i = 1, 2, \ldots, j = 1, 2, \ldots, n$, where $\{\{X_j^{(i)}\}_{j=1}^{n}\}_{i=1}^{\infty}$ are independent and identically distributed random variables (i.i.d. r.v.'s). At this time, the execution time of Phase i is $T_n^{(i)} = \max(X_1^{(i)}, X_2^{(i)}, \ldots, X_n^{(i)})$ because the execution time of the phase is the execution time of the slowest processor. Therefore, the execution time of m phases with n processors before eliminating barriers is $B_n^{(m)} = T_n^{(1)} + T_n^{(2)} + \cdots + T_n^{(m)}$ and the mean execution time becomes $E(B_n^{(m)}) = mE(T_n^{(1)})$.

After Eliminating Barriers Partially. In this paper, we use the term *dependent on part* to represent a situation that a processor depends on several processors rather than all other processors. Dependent on part situations appear in many parallel programs. On the other hand, the term *dependent on all* represents a situation that a processor depends on all other processors. The case of 'before eliminating barriers' we showed in Sect. 2.1 is dependent on all throughout all phases.

In general, before Processor j proceeds Phase i in dependent on all situation, it has to wait for the finish of Phase $(i-1)$ performed by all other processors. On the other hand, in dependent on part situation, Processor j has to wait for the finish of Phase $(i-1)$ performed by not all but several processors as shown in Fig. 1(b). In this paper, we call *depended processors* of Processor j in Phase i a group of the processors for which Processor j has to wait at the beginning of Phase i. We denote by $S_j^{(i)}$ the set of depended processors' IDs. Note that about $S_j^{(i)}$:

- $j \in S_j^{(i)}$ because Processor j always depends on itself.
- $S_j^{(1)}$ is empty set for all j because there is no memory access before Phase 1.

We also denote by $X_j^{(1,i)}$ the execution time of Processor j from the beginning of a program, that is Phase 1, to the end of Phase i. $X_j^{(1,i)}$ is decomposed as follows:

$$X_j^{(1,i)} = \max\left(X_k^{(1,i-1)}\right)_{k \in S_j^{(i)}} + X_j^{(i)}, i = 2, 3, 4, \ldots$$

In other words, $X_j^{(1,i)}$ is the sum of the longest execution time of depended processors at the end of Phase $(i-1)$ and the execution time of Phase i performed by Processor j.

We also denote the execution time of the program in which all processors execute by $A_n^{(m)}$. Then, $A_n^{(m)} = \max(X_1^{(1,m)}, X_2^{(1,m)}, \ldots, X_n^{(1,m)})$, that is, the random variable $A_n^{(m)}$ represents the execution time of m phases with n processors after eliminating barriers partially.

2.2 The Definition of Dependency Matrix

To represent partial dependency, it is necessary to describe depended processors, that is, the group of processors for which a processor has to wait at the beginning of a phase. To achieve this, we propose a *dependency matrix* in this paper as follows:

$$\mathbf{D} = \begin{pmatrix} \mathbf{d}_{11} & \mathbf{d}_{12} & \ldots & \mathbf{d}_{1n} \\ \mathbf{d}_{21} & \mathbf{d}_{22} & \ldots & \mathbf{d}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{d}_{m1} & \mathbf{d}_{m2} & \ldots & \mathbf{d}_{mn} \end{pmatrix}.$$

The element of matrix $\mathbf{D}$ is a binary vector $\mathbf{d}_{ij}$. The size of $\mathbf{d}_{ij}$ is equal to the number of processors, that is, $|\mathbf{d}_{ij}| = n$. Each element of $\mathbf{d}_{ij}$ represents dependency among processors. More specifically,

$$k\text{-th element of } \mathbf{d}_{ij} = \begin{cases} 1, & \text{if } k \in S_j^{(i)} \\ 0, & \text{if } k \notin S_j^{(i)}. \end{cases}$$

If k-th element of $\mathbf{d}_{ij}$ is 1, Processor j has to wait for Processor k at the beginning of Phase i.

The following is an example of $\mathbf{d}_{ij}$:

$$\mathbf{d}_{35} = (0, 1, 0, 0, 1, 1).$$

In this example, the number of processors is 6 due to $|\mathbf{d}_{35}| = 6$. $\mathbf{d}_{35}$ represents that Processor 5 has to wait the end of Phase 2 performed by Processor 2, 5, and 6 at the beginning of Phase 3. Because Processor 5 has to wait for itself, the fifth element of $\mathbf{d}_{i5}, i = 1, 2, \ldots, m$ is always 1.

Given a $\mathbf{D}$ and $\{\{X_j^{(i)}\}_{j=1}^{n}\}_{i=1}^{m}$, one can calculate an execution time $A_n^{(m)}$.

2.3 Probability Distribution of the Execution Time

The performance of barrier elimination may vary significantly depending on the distribution of random variables which represent the execution time for each processor and each phase. In this study, we assume that a random variable follows one of three probability distributions: exponential distribution, Erlang distribution, and hyper-exponential distribution.

In exponential distribution whose parameter is λ, the CDF (cumulative distribution function) for $X_j^{(i)}$ is assumed in the form

$$F(x) = P(X_j^{(i)} \le x) = 1 - e^{-\lambda x}$$

Table 1. Coefficients of variation (CV)

E_{100}	E_4	E_2	M	H_2(a) $\lambda_1 = 5\lambda$	H_2(b) $\lambda_1 = 10\lambda$	H_2(c) $\lambda_1 = 100\lambda$
0.1000	0.5000	0.7071	1.0000	1.5100	1.6186	1.9850

for $i = 1, 2, \ldots$ and $j = 1, 2, \ldots, n$, so that the PDF (probability density function) is in the form $f(x) = \lambda e^{-\lambda x}$ and expectation value (mean execution time) becomes $E(X_j^{(i)}) = \frac{1}{\lambda}$.

The CDF of Erlang distribution are

$$F(x) = 1 - e^{-\lambda k x} \sum_{r=0}^{k-1} \frac{(\lambda k x)^r}{r!},$$

where k is the number of phases[1]. The expected value of random variables which follow the above Erlang distribution is also $\frac{1}{\lambda}$. We chose k as $k = \{2, 4, 100\}$.

The CDF of hyper-exponential distribution are

$$F(x) = 1 - \sum_{j=1}^{k} C_j e^{-\lambda_j x},$$

where $\{C_j\}_{j=1}^{k}$ is an arbitrary discrete distribution. We chose these parameters as follows so that $E(X_j^{(i)}) = \frac{1}{\lambda}$: $(k, C_1, C_2, \lambda_1, \lambda_2) = (2, \frac{1}{2}, \frac{1}{2}, 5\lambda, \frac{5}{9}\lambda), (2, \frac{1}{2}, \frac{1}{2}, 10\lambda, \frac{10}{19}\lambda), (2, \frac{3}{5}, \frac{2}{5}, 100\lambda, \frac{400}{994}\lambda)$.

We denote exponential distribution, Erlang distribution, and hyper-exponential distribution by M, E_k, and H_k, respectively, derived from Kendall's notation in queuing theory.

The adoption of these distributions for the execution time is based on the following idea. For non-negative random variables with the same expectation value, the coefficient of variation (CV) is the most useful and popular characteristic parameter for comparing the degree of variation. The CV $c(X)$ for non-negative random variable X is defined by $c(X) = \sqrt{V(X)}/E(X)$, where $V(X)$ is variance of X, i.e., $V(X) = E(X^2) - E(X)^2$. It is clear that for fixed value of $E(X)$, as increases the value of $c(X)$, the variance of X also increases. In the field of probability theory, M, E_k, and H_k are the most typical distribution with different CV. It is well known that $c(X) = 1$ for M, $c(X) < 1$ for E_k, and $c(X) > 1$ for H_k. In other words, for the same value of expectation, E_k shows lower variance and H_k shows higher variance comparing with M.

Table 1 shows CVs for the seven distributions. Hereafter, we assume that $\lambda = 1$ without loss of generality.

[1] The term of *phase* which is used in the context of probability theory is unrelated to a phase which is included in parallel program.

3 Evaluation

To investigate how much impacts brought by the differences of the load balance among processors in executing parallel programs on the effect of the barrier eliminating algorithm, we calculated the execution time for three kinds of dependency patterns with varying the coefficient of variation.

We calculated the execution time using Monte Carlo simulation. We varied the number of phases $m = \{2, 3, \ldots, 10\}$ as well as the number of processors $n = \{2, 4, 8, 16, 32\}$. We made three kinds of typical dependency patterns (DPs) and stored them in dependency matrices $\mathbf{D}$. The details of each $\mathbf{D}$ are described later.

We performed five simulations for a pair of (m, n) with the initial number of steps N, that is, 10^3. We consider that the simulations are *successful* if all the five execution times are equal when the times are rounded off to the second decimal place. If the rounded times are not equal, we increase N by ten, and then perform five simulations again. Consequently, we observed that 45 simulations for all pairs of (m, n) are successful before N reached 10^{10}.

3.1 Results

In DP-1, a processor depends on neighbors and itself. More exactly, Processor j depends on Processor $j-1$, j, and $j+1$. Exceptionally, Processor 1 depends on Processor 1 and 2 as well as Processor n depends on Processor $n-1$ and n. This dependency pattern appears in some applications including image processing and physics simulation, in which the value of a point is computed using neighboring points. The part of $\mathbf{D}$ for $n = 4$ used in DP-1 is as follows:

$$\mathbf{D} = \begin{pmatrix} (0,0,0,0) & (0,0,0,0) & (0,0,0,0) & (0,0,0,0) \\ (1,1,0,0) & (1,1,1,0) & (0,1,1,1) & (0,0,1,1) \\ (1,1,0,0) & (1,1,1,0) & (0,1,1,1) & (0,0,1,1) \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}.$$

Dependency is fixed as phase ID increases.

Table 2 show the execution time for E_{100} and H_2(a). Due to limitations of space, the results for three other distributions are not shown but their values take place between E_{100} and H_2. As mentioned above, the expected value of the execution time for a single phase is 1 $(=\frac{1}{\lambda})$. For $n = 2$, the DP is identical to the pattern in the case that all barriers are remained. Furthermore, with respect to E_2, M, and H_2(a), the execution time obtained by simulation for $n = 2$ is identical to the execution time obtained by mathematical analysis shown in Yonezawa et al. [5], in which it is assumed that a random variable follows one of E_2, M, and H_2(a). Therefore, this imply the correctness of our simulation.

Table 2. The execution time (DP-1)

	n (E$_{100}$)					n (H$_2$(a))				
m	2	4	8	16	32	2	4	8	16	32
2	2.11	2.19	2.26	2.31	2.36	3.32	4.82	6.41	8.01	9.57
3	3.17	3.28	3.36	3.43	3.49	4.98	7.06	9.10	11.04	12.88
4	4.23	4.36	4.46	4.54	4.61	6.64	9.31	11.76	14.01	16.10
5	5.28	5.45	5.56	5.65	5.73	8.30	11.57	14.42	16.96	19.26
6	6.34	6.53	6.66	6.77	6.85	9.96	13.82	17.07	19.89	22.40
7	7.39	7.62	7.77	7.88	7.97	11.62	16.07	19.73	22.81	25.51
8	8.45	8.70	8.87	8.99	9.09	13.28	18.33	22.38	25.72	28.61
9	9.51	9.79	9.97	10.10	10.20	14.94	20.58	25.04	28.63	31.70
10	10.56	10.87	11.07	11.21	11.32	16.60	22.83	27.70	31.53	34.77

In DP-2, all processors depend on a single specific producer. It is also known as 'single writer, multiple reader'. This pattern appears in master-worker type applications. The part of $\mathbf{D}$ for $n = 4$ used in DP-2 is as follows:

$$\mathbf{D} = \begin{pmatrix} (0,0,0,0) & (0,0,0,0) & (0,0,0,0) & (0,0,0,0) \\ (1,0,0,0) & (1,1,0,0) & (1,0,1,0) & (1,0,0,1) \\ (1,0,0,0) & (1,1,0,0) & (1,0,1,0) & (1,0,0,1) \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}.$$

As in DP-1, dependency is fixed as phase ID increases.

DP-3 is similar to DP-2 since all processors depend on a single producer. However, producer's ID is incremented when processors go to the next phase in DP-3. If Processor n is the producer in a phase, Processor 1 becomes the next producer in the next phase. This pattern appears in some applications including Gaussian elimination method in which the rows of matrix is assigned to processors in block-cyclic manner. The part of $\mathbf{D}$ for $n = 4$ used in DP-3 is as follows:

$$\mathbf{D} = \begin{pmatrix} (0,0,0,0) & (0,0,0,0) & (0,0,0,0) & (0,0,0,0) \\ (1,0,0,0) & (1,1,0,0) & (1,0,1,0) & (1,0,0,1) \\ (1,1,0,0) & (0,1,0,0) & (0,1,1,0) & (0,1,0,1) \\ (1,0,1,0) & (0,1,1,0) & (0,0,1,0) & (0,0,1,1) \\ (1,0,0,1) & (0,1,0,1) & (0,0,1,1) & (0,0,0,1) \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}.$$

Unlike DP-1 and 2, dependency is varied as phase ID increases.

3.2 Discussion

Figure 2 shows the ratio of improvement which is calculated by $100 \times (1 - T_p/T_b)$, where T_b is the execution time in the case that all barriers are remained and T_p is the execution time in DP-1, 2, or 3 for $m = 10$ and $n = 8, 32$.

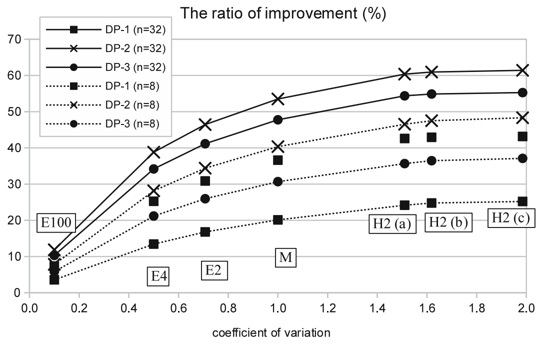


Fig. 2. The ratio of improvement ($m = 10; n = 8, 32$)

The common trends among three kinds of DPs are (1) if CV increases, namely, if the loads among processors is more imbalanced, the ratio of improvement also increases, (2) if n, the number of processors, increases, the ratio of improvement also increases. These trends are also observed in Yonezawa et al. [5]. For example, the ratio of improvement in the case that all random variables follow E_{100} is 11.87 % for $n = 32$ in DP-2 while the ratio in the case that all random variables follow H_2(c) is 61.41 %. Since the program contains much idle time for H_2, once barriers are partially eliminated, the opportunities to move the execution of phases forward increases. This decreases idle time and therefore increases the ratio of improvement. If n increases, the deviation of the execution time among processors increases. This causes idle time if all barriers are remained. Eliminating several barriers mitigates the deterioration for larger n.

The ratio of improvement in DP-2 and 3 are greater than DP-1. It is caused by the number of depended processors. In general, when the number of 1s included in dependency matrix **D** increases, the number of depended processors also increases. In this evaluation, the number of 1s in **D** for DP-2 is equal to the one for DP-3 while **D** for DP-1 has more 1s than two other DPs.

Whereas the number of 1s in **D** is the same between DP-2 and 3, the execution times differ. In DP-3, a producer in a phase depends on the previous producer directly. However, there are depended producers which influence indirectly from the past phases. This makes a chain of dependency across phases and the producer in Phase $j, j > n$, depends on all processors directly or indirectly.

DP-1 spends more time compared with DP-2 and 3. It is caused by the number of chains of dependency. In DP-1, a processor depends on two neighbors. Therefore, two chains are generated for the processor. This brings rapid growth of the number of depended processors indirectly.

In this paper, we consider an imaginary *optimal* situation, where all barriers are removed and all phases are performed consecutively with disregarding dependencies among processors. Consequently, the execution time include no idle time in the situation and is regarded as the upper bound of the effect of barrier eliminating method. We call the relative execution time *optimal degree*,

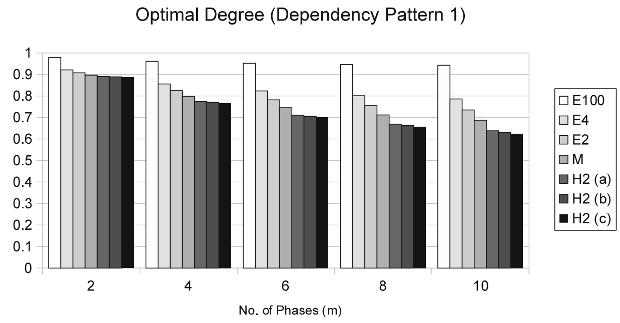


Fig. 3. Optimal degree (DP-1: $n = 32$)

which are calculated based on the optimal execution time. Although the value of optimal degree can be affected by the kinds of DPs and CVs, due to limitations of space, we show the result with varying CV while fixing DP. One can consider that a CV is near to optimal if the optimal degree is close to 1. Figure 3 shows optimal degree for $n = 32$ in DP-1. While we omit the results for two other DPs, optimal degree tends to decrease when CV increases in three DPs. For example, optimal degree in the case that all random variables follow E_{100} is 0.94 in DP-1 while optimal degree in the case that all random variables follow H_2(c) is 0.63. With E_{100}, the idle time are inherently short even if all barriers are remained due to a good load balance. In contrast to E_{100}, H_2(c) causes long idle time, which cannot be removed even after several barriers are eliminated.

Figure 4 shows speedup for $m = 10$ in DP-1, which is calculated based on $mn(= \frac{mn}{\lambda})$, that is, the execution time for uniprocessor. Speedup tends to decrease when CV increases. For example, the maximum speedup is 28.27 for $n = 32$ in DP-1 when all random variables follow E_{100}. In contrast, the speedup is limited to 6.73 for H_2(c).

From the comparison among the results of DP-1, 2, and 3, we found that the obtained effect of eliminating barriers is higher if a processors depends on

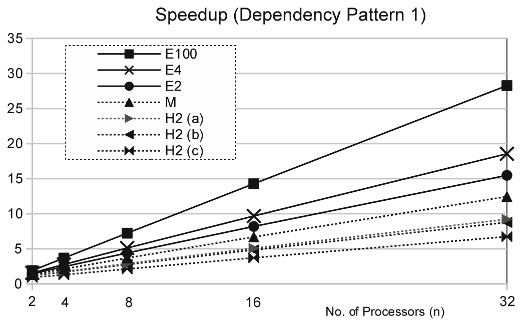


Fig. 4. Speedup (DP-1: $m = 10$)

less other processors. Even if a processor depends on few other processors, the excess barriers force the processor to wait for other non-depended processors. If a processor depends on quite a few other processors as in DP-3, the effect of eliminating barriers is limited due to the necessary interprocessor communications even after eliminating barriers. With regard to the load balance, we found that a better speedup is obtained if a load balance is better while the less effect of eliminating barrier is obtained. If loads among processors are imbalanced as in H_2, barrier elimination can contribute to improve the performance of parallel programs. These observations are consistent to the experimental results we have executed on PC clusters.

4 Related Work

There are several methods [1,2] to eliminate barrier synchronizations besides the method [3,4] we investigate in this paper. In both methods, however, some theoretical model is not constructed and no mathematical evaluation is carried out.

Sun and Peterson [6] represent the execution time by using random variables as in this paper. They focus on a method to approximate the execution time while we aim to optimize parallel programs by eliminating barriers partially.

5 Conclusion

In this paper, we proposed a probabilistic model which describes the behavior of a parallel program and measured the effect of the algorithm of barrier elimination. In our model, random variables show the execution time for one phase in which one processor performs. By introducing dependency matrix, we extended the model we have proposed in our previous work so that our extended model represents dependencies among processors.

For evaluation, we executed three kinds of parallel program in simulation. In order to investigate how a load balance influences the effect of barrier eliminating method, we adopted three probability distributions, that is, exponential distribution, Erlang distribution, and hyper-exponential distribution. We obtained the ratio of improvement, the optimal degree, and the speedup. Based on these results, we found that a better speedup is obtained if a load balance is better while the less effect of eliminating barrier is obtained.

In the future, we plan to make our study more accurate by sampling execution times of processors in runtime of real applications and then applying the samples to our model.

References

1. Dwarkadas, S., Cox, A.L., Zwaenepoel, W.: An integrated compile-time/run-time software distributed shared memory system. In: Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 186–197 (1996)

2. Tseng, C.W.: Compiler optimizations for eliminating barrier synchronization. In: Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, July 2005, pp. 144–155 (1995)
3. Yonezawa, N., Wada, K.: Reducing idle times on matrix programs by eliminating barrier synchronization. IEICE J. **J91-D**(4), 907–921 (2008)
4. Yonezawa, N., Wada, K., Aida, T.: Barrier elimination based on access dependency analysis for OpenMP. In: Guo, M., Yang, L.T., Di Martino, B., Zima, H.P., Dongarra, J., Tang, F. (eds.) ISPA 2006. LNCS, vol. 4330, pp. 362–373. Springer, Heidelberg (2006)
5. Yonezawa, N., Kino, I., Wada, K.: Probabilistic analysis of time reduction by eliminating barriers in parallel programmes. Int. J. Commun. Netw. Distrib. Syst. **6**(4), 404–419 (2011)
6. Sun, J., Peterson, G.D.: An effective execution time approximation method for parallel computing. IEEE Trans. Parallel Distrib. Syst. **23**(11), 2024–2032 (2012)

Multi-GPU Parallel Memetic Algorithm for Capacitated Vehicle Routing Problem

Mieczysław Wodecki[1], Wojciech Bożejko[2](✉), Michał Karpiński[1], and Maciej Pacut[1]

[1] Institute of Computer Science, University of Wrocław, Joliot-Curie 15, 50-383 Wrocław, Poland
{mwd,michal.karpinski,maciej.pacut}@ii.uni.wroc.pl
[2] Institute of Computer Engineering, Control and Robotics, Wroclaw University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland
wojciech.bozejko@pwr.wroc.pl

Abstract. The goal of this paper is to propose and test a new memetic algorithm for the capacitated vehicle routing problem in parallel computing environment. In this paper we consider a simple variation of the vehicle routing problem in which the only parameter is the capacity of the vehicle and each client only needs one package. We analyze the efficiency of the algorithm using the hierarchical Parallel Random Access Machine (PRAM) model and run experiments with code written in CUDA.

Keywords: Metaheuristics · Vehicle routing problem · GPGPU

1 Introduction

In Capacitated Vehicle Routing Problem (CVRP) we consider the following scenario: we own a delivery business that sends goods to clients via trucks. Transport begins at the base station. The time needed to travel from base station to every client (and from every client to every other client) is known. We can look at this set-up as a full weighted graph with one highlighted vertex. The goal is to deliver every package to clients in the smallest possible time according to their demands. The capacity of each truck is fixed. The truck needs to go back to the base station to reload when empty. A general CVRP assume that demand of every client, number of trucks and their capacity are not bound by any assertion. The vehicle routing problems have been attracting the interest of combinatorial optimization experts for over 50 years. The motivation to study this class of problems lies in its relevance to the real world as well as in its difficulty. One of books that are worth mentioning is [7]. It is an overview of main VRP variations (including CVRP). The authors show both exact and heuristic methods of

The work was supported by the OPUS grant DEC-2012/05/B/ST7/00102 of Polish National Centre of Science.

finding the solutions. Large portion of the book covers the main variations, like: VRP with time windows, backhauls, pickup and delivery.

In our variation of the CVRP we assume that every client demands exactly one package and we have only one delivery truck with fixed capacity. It is easy to see that with these constraints our problem transforms into a permutation problem. Furthermore, it is very similar to the classical Traveling Salesman Problem (TSP) [3] with only difference being exclusion of the base station from permutation. Therefore, only vertices that represent clients are being permutated. Next, we can evenly partition the resulting permutation into sets of size equal to the capacity of the truck. These sets represent paths the truck will make with each round of deliveries.

The similarity to the TSP doesn't end here. If we set the capacity of the truck to the number of clients, then CVRP becomes TSP. Because of that, further in the paper (in the experiments section), we test how well our algorithm performs in solving TSP problem on well known data sets taken from TSPLIB [8].

The memetic algorithm that we propose is the combination of Simple Genetic Algorithm [9] and Simulated Annealing [6]. It can be parallelized in a very natural way on multiple GPUs using the Island Model [2,4,10]. Each GPU contains one population. We apply series of genetic operators to the population. In addition, after each iteration of the algorithm, the local search algorithm is run on every specimen for further solution improvement. Thanks to the parallel nature of the GPU, we can apply all these functions to each of the specimen at the same time, which greatly accelerates the computation.

The rest of the paper is organized as follows: firstly, we give a specification of the CVRP variation we will be solving. Next, we introduce a new memetic multi-GPU algorithm and give its theoretical analysis of time complexity and cost. After that we show the results of the performed experiments. The main goal is to show the scalability of the algorithm.

2 Capacitated Vehicle Routing Problem

Let us consider a full weighted undirected graph $G = \langle V, E, w \rangle$, where $V = \{v_1, \cdots, v_n\}$ is the set of vertices, E is the set of edges, and $w : E \to \mathbb{N}$ is the weight function. Let v_1 be the base station. The problem is to find a disjoint full partitioning $X_m = \{X_1, \cdots, X_k\}$ of set $V_1 = V \setminus \{v_1\}$ and permutations σ_i for each X_i such that:

$$X_m = \underset{X \in \mathcal{P}}{\text{minarg}} \sum_{i=1}^{k} \left(\sum_{j=1}^{|X_i|-1} w(\sigma_i(j), \sigma_i(j+1)) + w(v_1, \sigma_i(1)) + w(\sigma_i(|X_i|), v_1) \right),$$

where $\mathcal{P}$ is a set of all valid partitionings. In our variation the size of each set X_i in partitioning is constrained and it is equal to c, which is the parameter of the problem. We can interpret c as the capacity of the truck. Note that with this constraint the truck always takes exactly c packages from the base station. Also note that since the partitioning X is disjoint, the truck never visits the same client twice and is only going back to the base station after the loading is empty.

The problem formulated in such way is NP-complete, which can be shown with the reduction from Minimum Assignment Problem [1].

3 The GPU Algorithm

To solve the capacited vehicle routing problem we chose memetic algorithm, which is a modification of a genetic algorithm by addition of local search in each iteration. The evolutionary operators used are: CX, OX, PMX [5]. Algorithm 1.1 presents a parallel version of the memetic algorithm.

Algorithm 1.1. Parallel memetic algorithm schema

```
parfor(1..g) do
  population = random_population();
  while(termination_conditon()) do
    parfor(1..c) do
      calculate_fitness(population);
      mutation(population);
      crossover(population);
      local_search(population);
      selection(population);
    end
    out_migration = select_migration(population);
    broadcast(out_migration);

    in_migration = receive();

    population.append(in_migration);
    selection(population);
  end
  return best(population);
end
```

3.1 Algorithm Analysis

We use the hierarchical PRAM model in the analysis. The hierarchy consists of CPU and its memory and multiple GPUs with their memory and cores. We assume that the number of islands is equal to number of GPUs, and the size of population on each isle is equal to the number of cores on GPU (all GPUs are the same).

Let's assume following denotations:

- $T(g, c, n, i)$ - time of execution of algorithm on g GPUs, each with c cores on data of size n on i iterations
- $C(g, c, n, i)$ - cost of execution of algorithm with argument named above. Equation for cost is $C(g, c, n, i) = g \cdot c \cdot T(g, c, n, i)$.

Execution time is influenced by following variables:

- n - number of vertices in graph
- g - number of islands (equal to number of GPUs)
- c - size of population on each isle (equal to number of cores)
- e - number of specimen sent to other islands on every migration
- i - number of performed iterations
- f - frequency of migrations
- $cross(n)$, $mut(n)$, $eval(n)$ - time costs of single crossover, mutation and evaluation
- pr_{cross}, pr_{mut} - probability of applying crossover and mutation

Time complexity is calculated as follows:

$$\begin{aligned}
T(g,c,n,i) &= T_{init} + i \cdot (T_{cross} + T_{mut} + T_{eval} + T_{i\_sel}) + \\
&\qquad + \frac{i}{f} \cdot T_{o\_sel}, \\
T_{init} &= O\left(\frac{c \cdot n}{c}\right), \\
T_{cross} &= O\left(\frac{pr_{cross} \cdot cross(n) \cdot \binom{c}{2}}{c}\right), \\
T_{mut} &= O\left(\frac{pr_{mut} \cdot mut(n) \cdot c}{c}\right), \\
T_{eval} &= O\left(\frac{eval(n) \cdot c}{c}\right), \\
T_{i\_sel} &= O\left(\log\left(c + \binom{c}{2} \cdot pr_{cross}\right)\right), \\
T_{o\_sel} &= O\left(\log(c + e \cdot (p-1))\right).
\end{aligned}$$

$T_{init}, T_{cross}, T_{mut}, T_{eval}$ are divided by c, because each GPU has c cores that can perform those operations in parallel. T_{i_sel} equals to the time of parallel sorting of population enlarged by the specimens that were created during crossover. T_{o_sel} equals to the time of parallel sorting of population enlarged by the specimens received from other islands. Costs of crossover, mutation and evaluation are at most linear with respect to n. From the above assumptions we have:

$$T(g,c,n,i) \in O(i \cdot (n + pr_{cross} \cdot n \cdot c) + \frac{i}{f} \cdot n \cdot (\log(c + e \cdot g))).$$

Assuming that the amount of exchanged specimens and the probability of mutation are usually small, we can treat those values as constants, resulting with:

$$T(g,c,n,i) \in O(i \cdot n + \frac{i}{f} \cdot n \cdot \log(c + g)).$$

With the above simplification we can estimate the cost as:

$$C(g,c,n,i) \in O(c \cdot (g \cdot i \cdot n + \frac{i}{f} \cdot g^2)).$$

We chose a model of the speedup where we require the sequential and parallel algorithm to perform the same number of iterations. For each iteration of the sequential algorithm, the parallel algorithm performs g times more iterations because it operates on g GPUs. To calculate the speedup we launch the sequential algorithm for $i \cdot g$ iterations and the parallel algorithm for i iterations on each GPU.

$$S(g, c, n, i) = \frac{T(1, c, n, i \cdot g)}{T(g, c, n, i)} = \frac{n \cdot (g \cdot i)}{i \cdot n + \frac{i}{f}(n \cdot \log g)} = \frac{n \cdot g}{n + \frac{1}{f}(n \cdot \log g)}.$$

For obtaining maximal performance n must be close to the number of cores, so we substitute $c = n$. The number of cores in one nVidia Tesla S2050 GPU is 448. The optimal frequency obtained from our experiments equals 50. Final form of the speedup is:

$$S(g, 448, 448, i) = \frac{448 \cdot g}{448 + \frac{1}{50}(448 \cdot \log g)} \tag{1}$$

4 Computational Experiments

The implementation of the algorithm was written in C++ using CUDA and OpenMPI library. All the tests were performed on nVidia Tesla S2050 1U Computing System.

The first batch of tests we executed on randomly generated data using one GPU. Our goal was to empirically determine the best parameters for our program. In Table 1 we can see the performance results of the basic crossover operators: PMX, OX and CX. We measure the performance in number of iterations the algorithm has to make in order to reach certain reference solution using one of the crossover operators. Total of 1000 different results were generated at this stage of experiments and the size of input data was increasingly larger. In over 99 % cases the CX operator yielded best results, as it had the fastest convergence rate. We selected a few most interesting results.

We picked the best value for probability of mutation in similar manner. This time we plotted 1000 graphs from execution of program run on randomly generated data. This time we only chose CX operator for crossover, paying attention to the previous results. We studied the plots for mutation probability values ranged

Table 1. Performance for crossover operators on single GPU (n - number of clients).

n	OX	PMX	CX
30	19	20	**10**
51	69	56	**32**
99	354	249	**101**
300	2510	1223	**467**
501	11559	5979	**234**
999	23043	7384	**1534**

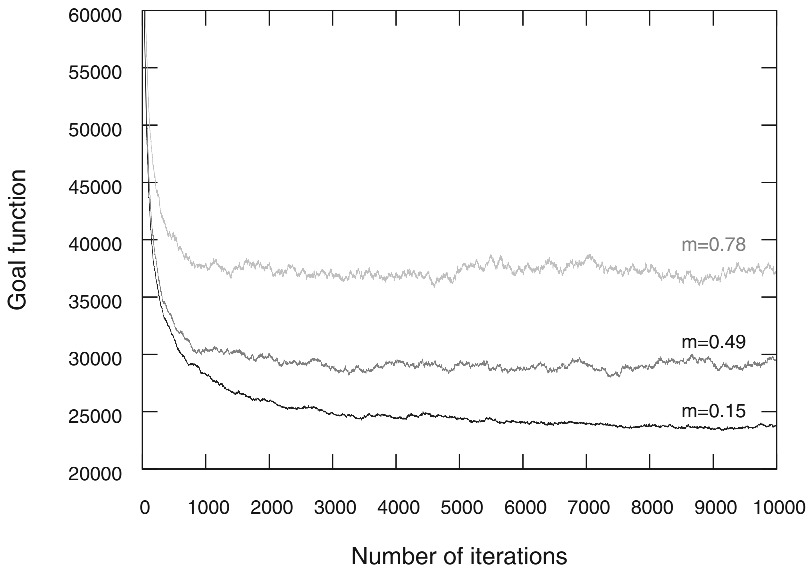


Fig. 1. Algorithm performance on single GPU with different mutation probabilities.

from 0.01 to 1.00. High values of pr_{mut} obviously produces random results. No signs of convergence was seen. We concluded, that the best value is $pr_{mut} = 0.15$. In Fig. 1 we present the selected plots (tests executed on very large input).

The data sets with the known optimal results for our CVRP variation does not exist, so it is hard to test the algorithms performance. To summarize our experiments we need to refer our results to some previous results. In order to make a comparison with the results of other researchers, we fixed the capacity of the truck to the number of cities. Now we could solve instances of the traveling salesman problem, so we were able to use the known data sets, which gives us at least some knowledge about the performance of the algorithm. In Table 2 we can see the results of this set of tests. We present percentage relative deviation of optimal solutions to selected problem instances taken from TSPLIB.

Table 2. Comparison of memetic algorithm to selected TSPLIB instances.

problem	*prd*	*problem*	*prd*
bayg29	0.00	bays29	0.00
brazil58	0.55	brg180	1.65
gr120	1.01	hk48	0.87
gr17	0.00	si1032	7.79
gr21	0.00	si175	1.48
gr24	0.00	si535	6.92
gr48	0.73	swiss42	0.24

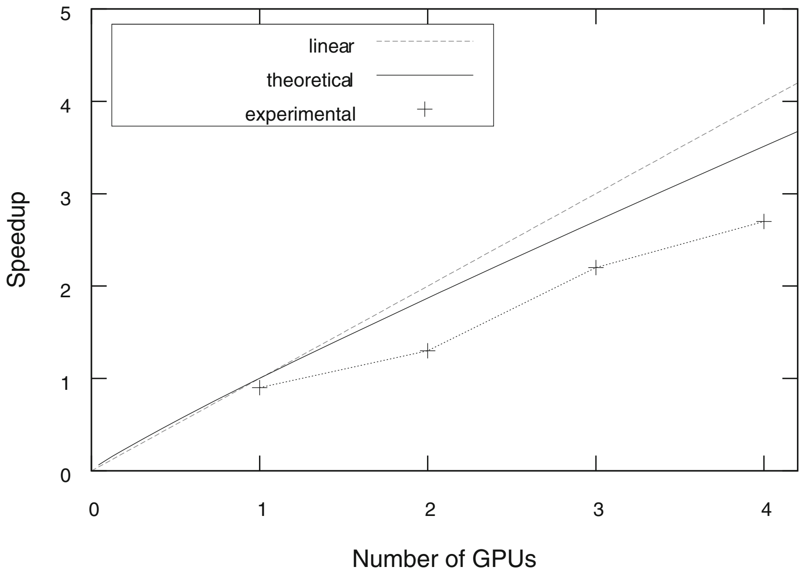


Fig. 2. Comparison of experimental and theoretical speedup.

Finally we show how the experimental speedup relates to the theoretical speedup (Eq. 1). We measured 1000 experimental speedups and plotted its average on the same graph that we plotted the theoretical speedup. Results are shown in Fig. 2.

5 Conclusions

Speedup results were satisfactory enough, but not ideal. We believe it's due to a very simple parallel model. Note that we could only test speedup for limited number of GPUs due to the fact that nVidia Tesla S2050 Computing System has only 4 GPUs. Either way, in Fig. 2 we can see that the experimental speedup behaves similarly as the theoretical speedup. On another note our memetic algorithm can be used in solving TSP as the results were not far away from optimal.

The vehicle routing problem and its many variations are NP-complete. It would seem that our variation might not be computionally hard, since we greatly simplified the constrains. Unfortunately our variation is NP-complete, which we show in different paper that is not yet published. We also explore more variations of CVRP adding more constraints to the problem discussed here. It may be reassuring that if we set the capacity of the truck to 2, the problem becomes trivially easy to compute. On the other hand, it might not even be practically usable. Either way, we believe that it is important to be aware of the barriers relaying in computing difficulty of the problem, which is another motivation to study simplified versions of the NP-complete problems.

References

1. Bandelt, H., Crama, Y., Spieksma, F.: Approximation algorithms for multi-dimensional assignment problems with decomposable costs. Discrete Appl. Math. **49**, 25–40 (1994)
2. Bożejko, W., Uchroński, M., Wodecki, M.: The new golf neighborhood for the flexible job shop problem, Proceedings of the ICCS 2010. Procedia Computer Science **1**, 289–296 (2010)
3. Bożejko, W., Wodecki, M.: On the theoretical properties of swap multimoves. Oper. Res. Lett. **35**(2), 227–231 (2007). (Elsevier Science Ltd.)
4. Bożejko, W., Wodecki, M.: Parallel genetic algorithm for minimizing total weighted completion time. In: Rutkowski, L., Siekmann, J., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 400–405. Springer, Heidelberg (2004)
5. Deepa, S.N., Sivanandam, S.N.: Introduction to Genetic Algorithms. Springer, Heidelberg (2008)
6. Gelatt, C.D., Kirkpatrick, S., Vecchi, M.P.: Optimization by simulated annealing. Sci. New Ser. **220**(4598), 671–680 (1983)
7. Toth, P., Vigo, D.: The Vehicle Routing Problem. Society for Industrial and Applied Mathematics, Philadelphia (2001)
8. TSPLIB data sets. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
9. Vose, M.D.: The Simple Genetic Algorithm: Foundations and Theory. Massachusetts Institute of Technology, Cambridge (1999)
10. Whitley, D.: A genetic algorithm tutorial. Stat. Comput. **4**(2), 65–85 (1994)

Parallel Applications Performance Evaluation Using the Concept of Granularity

Jan Kwiatkowski[✉]

Institute of Informatics,
Wroclaw University of Technology,
Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland
jan.kwiatkowski@pwr.wroc.pl

Abstract. With the advent of multi-core processors and the growing popularity of local clusters installations, better understanding of parallel applications behaviour becomes a necessity. On the other hand performance evaluation constitutes an intrinsic part of every application development process. The performance analysis can be carried out analytically or through experiments. When using experimental approach, its results are based on wall-time measurements and requires consecutive application executions which is time-consuming. In the paper an alternative approach is proposed. Utilizing the decomposition of execution time, a separate analysis of the computation time and overheads related to parallel execution are used to calculating the granularity of application and then determining the efficiency of the application. The usefulness of the new technique has been evaluated by comparing its results with those of classical ones. The obtained results suggest that the presented method can be used for performance evaluation of parallel applications.

Keywords: Performance evaluation · Application granularity

1 Introduction

Performance evaluation constitutes an intrinsic part of every application development process. In parallel programming the goal of the design process is not to optimise a single metrics like for example speed. A good design has to take into consideration memory requirements, communication cost, efficiency, implementation cost, and others. Therefore performance evaluation of parallel programs are very important for the development of correct and efficient parallel applications. In general the performance analysis can be carried out analytically or through experiments. In the first case it is call performance modelling, when in the second one performance measurement. A typical approach to performance measurement based on instrumenting the application, monitoring its execution and finally analysing its performance using collected information by some dedicated tool. The instrumentation can used some specific hardware features as for example the performance counters as well as architectural features such as trap or breakpoint instructions. On the other hand in performance measurement two different approaches can be distinguish, the first when the time measurement is included as additional code in the evaluated application and the second one when some specific external tool is used for time measurement, for example, profilers. Later, the second approach can be divided onto two classes of tools, the first which modified existing operating system and the second one which are autonomic. Independently on the used evaluation method, analytical or experimental the same performance metrics are used [2]. For parallel application performance evaluation various metrics are used - the run time, speedup or efficiency are all usually measured using the wall-clock time [3] and requires consecutive application executions which is time-consuming. In the papers [5,7,9] the methods, which overcome above problem for single core processors have been proposed.

In the paper an alternative approach that is extension of both methods mentioned above is proposed. It based on the concept of granularity and decomposition of the execution time between the time devoted to the computations and the overhead related to the parallel application execution and can be used for multi-core processors. The paper is organised as follows. Section 2 briefly describes different metrics used during parallel application performance evaluation. The idea of using granularity in performance evaluation as well as discussion related to components of parallel run time is presented in Sect. 3. The next section illustrates the experimental results obtained during evaluation of different parallel algorithms using proposed method and briefly describes these algorithms. The comparison of obtained results with results of standard methods is also included. Finally, Sect. 5 outlines the work and discusses ongoing work.

2 Performance Metrics

During performance evaluation of parallel applications different metrics are used [3,4,8]. The first one is the parallel run time ($t_{runtime}$). It is the time from the moment when computation starts to the moment when the last processor finishes its execution. The parallel run time is composed as an average of three different components: computation time, communication time and idle time. The computation time (t_{comp}) is the time spent on performing computation by all processors, communication time (t_{comm}) is the time spent on sending and receiving messages by all processors, the idle time (t_{idle}) is when processors stay idle. So, it can be described by the equation $t_{runtime} = (t_{comp} + t_{comm} + t_{idle})/p$, where p is a number of used processors. Moreover the computations time consists of the parallelized part of the sequential algorithm, additional computations that have to be performed in the parallelized version of the algorithm and the computations needed to deal with the communication (e.g. data marshalling): $t_{comp} = t_{comp\_single} + t_{comp\_ovhd\_comp} + t_{comp\_ovhd\_comm}$. The communication time t_{comm} is defined as the time which processors spend waiting for the data from other processors to arrive or (if the synchronous transfer is used) waiting for other parties to receive the data sent. The CPU time they spend on sending their data is already included in $t_{comp\_ovhd\_comm}$.

The parallel run time of a parallel algorithm depends not only on the size of the problem but also on the complexity of the interconnection network and the number of processors used. The next commonly used metric is speedup, which captures the relative benefit of solving a given problem using a parallel system. There exist different speedup definitions. Generally the speedup (S) is defined as the ratio of the time needed to solve the problem on a single processor to the time required to solve the same problem on a parallel system with p processors. Depending on the way in which sequential time is measured we can distinguish absolute, real and relative speedups. Theoretically, speedup cannot exceed the number of processors used during program execution, however, different speedup anomalies can be observed [6].

Both above mentioned performance metrics do not take into account the utilisation of processors in the parallel system. While executing a parallel algorithm processors spend some time on communicating and some processors can be idle. Then the efficiency (E) of a parallel program is defined as a ratio of speedup to the number of processors. In the ideal parallel system the efficiency is equal to one but in practice efficiency is between zero and one, however because of different speedup anomalies, it can be even greater than one. Similar to efficiency metrics is efficacy (η) that is a ratio of the speedup to square to the number of processors. When the efficacy has maximum value for some number of processors it means that this number of processors maximize application speedup per unit of cost. The next measure, which is often used in the performance evaluation of parallel programs, is the cost of solving a problem by the parallel system. The cost is usually defined as a product of the parallel run time and the number of processors. The next useful measure is the scalability of the parallel system. It is a measure of its capacity to increase speedup in proportion to the number of processors. We say that a system is scalable when the efficiency is the same for increasing the number of processors and the size of the problem [4].

Concluding the above short description of different performance metrics we can say that during experimental performance evaluation of parallel programs we need to measure the run time of sequential and parallel programs. However, there is a question: Is it possible to evaluate a parallel program using the above metrics by executing only a parallel version of the program on a parallel computer?

3 Using Granularity for Performance Analysis

In general the granularity of a parallel computer is defined as a ratio of the time required for a basic communication operation to the time required for a basic computation operation and for parallel algorithms as the number of instructions that can be performed concurrently before some form of synchronisation needs to take place. Let's defined the granularity of the parallel application similarly to above definition for parallel computers as the ratio of the amount of computation to the amount of communication within a parallel algorithm implementation ($G = T_{comp}/T_{comm}$). This definition can be used for calculating the granularity of a single process executed on the single processor as well as for the whole program using total communication and computation times of all program processes. Let us to calculate parallel program efficiency using the above definition of granularity. For this aim we defined the overhead function, which determines all overheads in the parallel algorithm compared with the best serial algorithms.

The overhead function is a function of problem size and the number of processors and is defined as follows [4]:

$$\mathcal{T}_o(W,p) = p * T_p - W \tag{1}$$

where W denotes the problem size, T_p denotes time of parallel program execution and p is the number of used processors. The problem size is defined as the number of basic computation operations required to solve the problem using the best serial algorithm. Let us assume that a basic computation operation takes one unit of time. Thus the problem size is equal to the time of performing the best serial algorithm on a serial computer. Based on the above assumptions after rewriting the Eq. (1) we obtain the following expression for parallel run time:

$$\mathcal{T}_p = \frac{W + T_o(W,p)}{p} \tag{2}$$

Then the resulting expression for efficiency takes the form:

$$\mathcal{E} = \frac{1}{1 + \frac{T_o(W,p)}{W}} \tag{3}$$

Recalling that the parallel run time consists of computation time, communication time and idle time. If we assume that the main overhead of parallel program execution is communication time then Eq. (3) can be rewritten as follows:

$$\mathcal{E} = \frac{1}{1 + \frac{T_{total\_comm}}{W}} \tag{4}$$

The total communication time is equal to the sum of the communication time of all performed communication steps. Assuming that the distribution of data among processors is equal then the communication time can be calculated using equation $T_{total\_comm} = p * T_{comm}$. Note that the above is true when the distribution of work between processors and their performance is equal. Similarly, the computation time is the sum of the time spent by all processors performing computation. Then the problem size W is equal to $p * T_{comp}$.

Finally, substituting the problem size and total communication time in Eq. (4) by using above equations we get:

$$\mathcal{E} = \frac{1}{1 + \frac{T_{comm}}{T_{comp}}} = \frac{1}{1 + \frac{1}{G}} = \frac{G}{G+1} \tag{5}$$

It means that using the concept of granularity we can calculate the efficiency and speedup of parallel algorithms.

Taking into consideration the brief discussion related to the parallel run time from previous section we can determined that in the above consideration not all overheads that appears during parallel program execution are considered. Moreover during experiments the measurements are performed mainly for the computations time (t_{comp}) and the wall-clock time (t_{wall}) that represents parallel run time. Then when computational overheads can be neglected all determined overheads can be expressed by: $t_{overhead} = t_{wall} - t_{comp}$. Then during performance measurement instead of granularity isogranularity defined as ($G_{iso} = t_{comp}/t_{overhead}$) will be used.

Concluding above consideration it is possible to evaluate a parallel application using such metrics as efficiency and speedup by measuring only the computation and wall-clock times during execution of parallel version of a program on a parallel computer.

4 Case Studies

To confirm the usefulness of the theoretical analysis presented in the previous section the series of experiments were performed. During the experiments three different algorithms were used: K-means, solving longest common subsequence problem (LCS) and Cannon Matrix Multiplication. The tests were executed on cluster located at the Institute of Informatics, Wroclaw University of Technology. The cluster is equipped with 13 homogeneous IBM HS21 with Xeon Quad Core X5365 - 3.0 GHz, space-shared nodes connected with the fast Ethernet network switch. To avoid the execution time anomalies [6] the experiments were performed for data sizes sufficiently larger than CPU cache size and smaller than the main memory limits. Because the experiments were performed in a multi-user environment the execution times depended on computer load, therefore the presented results are the averages from the series of 10 identical experiments performed. Moreover the results of measurement lying in the distance above 1.5 interquartile range of the whole series were treated as erroneous and omitted, and the measurement was repeated. To evaluate the accuracy of the new method the relative error defined as $\frac{\mathcal{S}-S}{S}$ where S is the actual speedup and $\mathcal{S}$ is the estimated one is presented for each tested algorithm.

K-means is one of the algorithms that is used for solving the clustering problem [10]. It classifies a given data set (for example point in 2-dimension space) into defined fixed number of clusters k (predefined). In the first algorithm's step so called centroids for each cluster should be chosen - one for each cluster. These centroids can be defined in random way however the better choice is to place them as much as possible far away from each other. In the next step all points from the data set are assign to the nearest centroid. After completion of this step the new centroids for each cluster are calculated using the means metrics for the created clusters. Then we repeated the second step using these new centroids. The process is continue as long as the differences between coordinates of new and old centroids are satisfied. Alternatively the process can be finished after predefined number of iterations. Concluding algorithm aims at minimizing some defined objective function, for example a squared error function.

The above algorithm was parallelized in the following simple way. The chosen processor reads input data, and then distributes them to other processors. Each processor received N/p data, when p is a number of available processors and N is the number of input data. Then each processor generates the appropriate number of centroids (it depends on the number of received data, and number of predefined clusters) and exchanges information about them with other processors. After completion of above step each processor has information about all the centroids and performs the second step of the sequential algorithm. In the next step each processor calculates the data necessary to calculate new centroids (the number of point in each cluster and sums of points coordinates) and exchange this information with other processors. Then the new centroids are calculated, in parallel by all processors (execution replication), and again the process returns to the second step of sequential algorithm. The algorithm ends when the stop criterion is met. Then the chosen processor collects clustering results from other processors and merge them.

The next algorithm solves a problem to find the length of the longest common subsequence of two sequences A and B and it bases on matrix $LCS_{|A|\times|B|}$. Let $lcs_{k,g}$ be an element of matrix LCS and it is the length of the longest common subsequence for the first k elements from A and g first elements from B [1]. The solution is given by the recurrence formulation:

$$lcs_{k,g} = \begin{cases} 0 & \text{for } k < 1 \text{ or } g < 1 \\ lcs_{k-1,g-1} + 1 & \text{for } k, g \geq 1 \text{ and } A[k] = B[g] \\ max(lcs_{k-1,g}, lcs_{k,g-1} & \text{for } k, g \geq 1 \text{ and } A[k] \neq B[g] \end{cases} \tag{6}$$

The computation goes from $k = 1$ and $g = 1$ to $k = |A|$ and $g = |B|$. We can see from formulation, that to compute value for pair k, g we neeed only maximum three value computed earlier, which are very near in the matrix.

In the parallel case a strip of the matrix with size $|A| \times (|B|/p)$ is assigned to each processor. The first processor computes m_s rows of its part of submatrix and then send the right most result to second processor. The second processor starts its computation of this same number of rows, and right result sends to next processor. In this same time the first processor computes the next m_s rows in its submatrix. This mean, that processors compute in pipeline and for one processor there are $|A|/m_s$ messages sent and received. In whole computation each processor send (except last one) and receive (except first one) a whole column of data of length $|A|$.

The last algorithm Cannon Matrix Multiplication algorithm as the classical parallel algorithm has been implemented according with its definition [4].

4.1 Experimental Results

In the experiments performed for K-means algorithm data set sizes 20000, 100000 and 200000 were used. The number of generated clusters was 20, 100 and 200, respectively. Moreover different hardware configurations by means different number of cores from each processor were used. Received results are presented on Figs. 1, 2, 3, and 4.

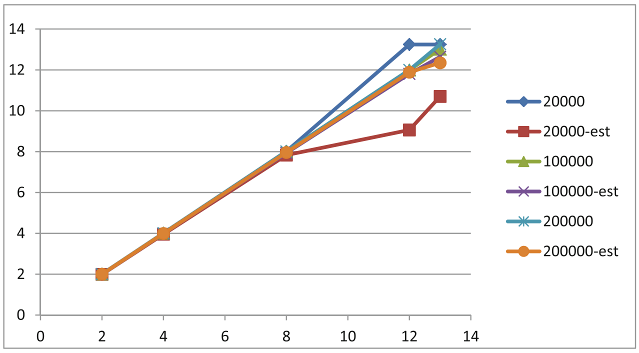


Fig. 1. K-means algorithm speedup and estimated speedup - 1 core at each node

The first test was performed using 1 core from 1, 2, 4, 8, 12, and 13 processors, its results are presented on Fig. 1. As can be seen the actual speedup and estimated speedup are very close, only when using 12 processors and the size of data set is equal 20000 there are large differences between actual and estimated speedup and the precise relative error is over 20 %, when for other cases between 0,2 % and 5 %.

In the second test 2, 4, 8, 16, 24 and 26 processing units (cores), two from each processor were used. Results of this test are presented on Fig. 2. As previously can be seen that the actual speedup and estimated speedup are very close. The difference appears when using 12 processors and the size of data set is equal 20000 as previously. However for this test the precise relative error values are slightly larger, between 0,2 % and 6 % and over 30 % for the first case.

In the third test 4, 8, 16, 24, 48 and 52 processing units (cores), four for each processor were used. Results of this test are presented on Fig. 3. In this test results are satisfied, too however for case when the size of the data set is equal

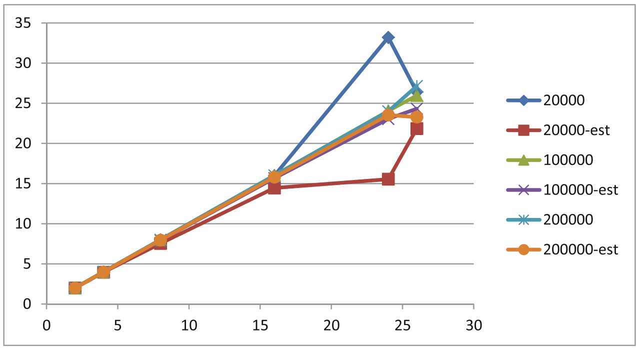


Fig. 2. K-means algorithm speedup and estimated speedup - 2 cores at each node

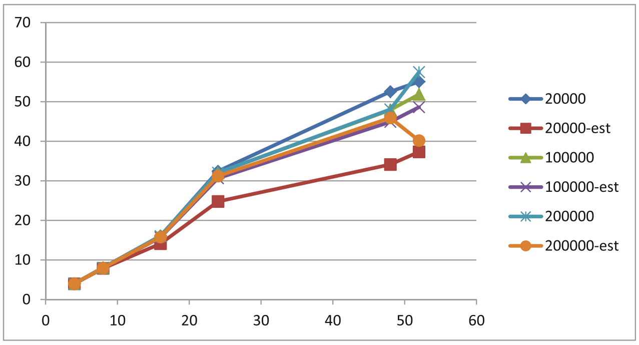


Fig. 3. K-means algorithm speedup and estimated speedup - 4 cores at each node

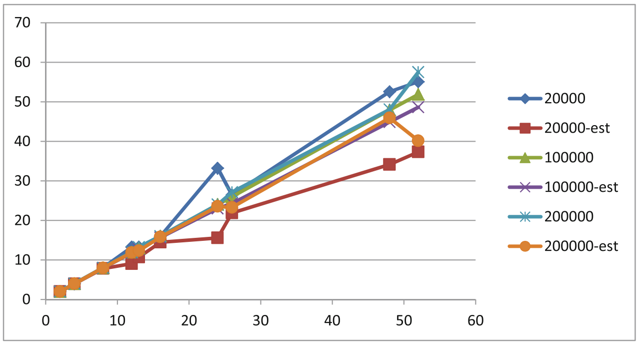


Fig. 4. K-means algorithm speedup and estimated speedup

20000 the precise relative error is much larger, between 1 % and 30 %, when for other cases between 0,4 % and 6 %.

In the last test performed for K-means algorithm 2, 4, 8, 12, 13, 16, 24, 26, 48 and 52 processing units (cores) randomly chosen from 52 available at cluster were used. As during the previous tests the worst results are for size of data set equals 20000, the precise relative error values are between 0,8 % and 30 %. In two other cases are between 0,2 % and 6 %.

The experiments for the LCS algorithm were performed for the constant size of both strings $|A| = |B| = 10000$. The m_s sizes of 4, 5, 10, 100 and 1000 were used for test. The measurements were performed for 1, 4, 9, 16 and 25 processing units (cores randomly chosen from 52 available at cluster).

The Fig. 5 presents the application actual and the estimation speedup values. Although the shapes of the functions are similar, even at the first glance it can be noted that in the case of the smaller m_s values the estimated speedups are shifted upwards comparing to the actual results and the precise relation error is even over 100 %, when for larger m_s is between 0 % and 7 %.

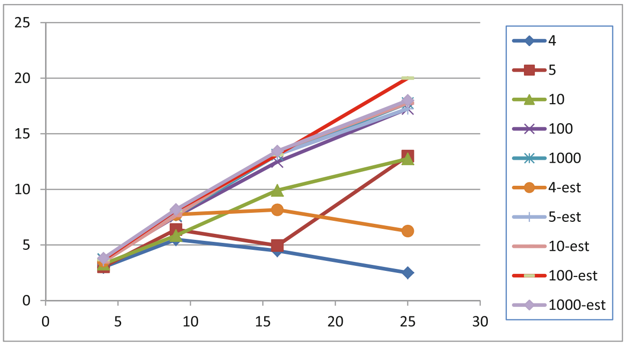


Fig. 5. LCS algorithm speedup and estimated speedup

In the experiments performed for the Cannon Matrix Multiplication algorithm the matrix sizes 504*504, 840*840 and 1008*1008 were used. The measurements were performed for 1, 4, 9, 16 and 25 processing units (cores) randomly chosen from 52 available at cluster.

On the Fig. 6 the actual and estimated speedups are presented. It can be seen that the estimated speedups closely match the actual ones for the larger matrix than for the smaller one. The precise relative error values are between 1 and 10

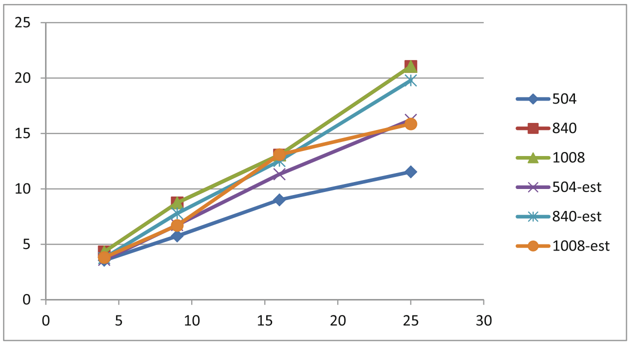


Fig. 6. Cannon Matrix Multiplication speedup and estimated speedup

5 Conclusions and Future Work

The paper propose a new way of calculating speedup and efficiency of parallel algorithms. The method is based on the idea of granularity and makes it possible to calculate the efficiency and speedup of parallel algorithm by executing only the parallel version of a program on a parallel computer. The method requires only the readily available data, without the need of installation of additional software or application modifications, we need only to measure wall-clock time and computational time. The experiments performed proved that the estimation accuracy is sensitive for different hardware structures when using multicore processors. For all analysed algorithms the results obtained are similar: the shape of diagrams is similar and the value of speedup is close. When the additional computational overhead is present, the accuracy is reduced.

The project is in current study, the first received results are very promising. Further research should investigate the possibility of using the isogranularity for evaluation algorithms with speedup anomalies, as well as the possibility of using isogranularity for scalability analysis.

References

1. Alves, C.E.R., Cceres, E.N., Song, S.W.: Sequential and parallel algorithms for the all-substrings longest common subsequence problem. Universidade de So Paulo, Instituto de Matemtica e Estatstica (2003)
2. Cremonesi, P., Rosti, E., Serazzi, G., Smirni, E.: Performance evaluation of parallel systems. Parallel Comput. **25**, 1677–1698 (1999). (North-Holland)
3. Foster, I.: Designing and Building Parallel Programs. Addison-Wesley, Reading (1995). (http://www.mcs.anl.gov/dbpp/text/book.html)
4. Grama, A.Y., Gupta, A., Kumar, V.: Isoefficiency: measuring the scalability of parallel algorithms and architectures. IEEE Parallel Distrib. Technol. **1**, 12–21 (1993)
5. Kwiatkowski, J.: Evaluation of parallel programs by measurement of its granularity. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2001. LNCS, vol. 2328, pp. 145–153. Springer, Heidelberg (2002)
6. Kwiatkowski, J., Pawlik, M., Konieczny, D.: Parallel program execution anomalies. In: Proceedings of the First International Multiconference on Computer Science and Information, Wisla, Poland (2006)
7. Kwiatkowski, J., Pawlik, M., Konieczny, D.: Comparison of execution time decomposition methods for performance evaluation. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 1160–1169. Springer, Heidelberg (2008)
8. Kumar, V., Grama, A., Gupta, A., Karypis, G.: Introduction to Parallel Computing. The Benjamin/Cummings Publishing Company Inc., Redwood City (1995)
9. Pawlik, M., Kwiatkowski, J., Koniecznym, D.: Parallel program performance evaluation with execution time decomposition. In: Proceedings of the 16th International Conference on Systems Science, Wroclaw, Poland (2007)
10. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Symposium on Mathematical, Statistics, and Probability, pp. 281–297. University of California Press, Berkeley (1967)

Workshop on Parallel Computational Biology (PBC 2013)

Resolving Load Balancing Issues in BWA on NUMA Multicore Architectures

Charlotte Herzeel[1,4(✉)], Thomas J. Ashby[1,4], Pascal Costanza[3,4], and Wolfgang De Meuter[2]

[1] imec, Kapeldreef 75, 3001 Leuven, Belgium
{charlotte.herzeel,ashby}@imec.be
[2] Software Languages Lab, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussel, Belgium
wdmeuter@vub.ac.be
[3] Intel, Veldkant 31, 2550 Kontich, Belgium
pascal.costanza@intel.com
[4] ExaScience Life Lab, Kapeldreef 75, 3001 Leuven, Belgium
http://www.exascience.com

Abstract. Running BWA in multithreaded mode on a multi-socket server results in poor scaling behaviour. This is because the current parallelisation strategy does not take into account the load imbalance that is inherent to the properties of the data being aligned, e.g. varying read lengths and numbers of mutations. Additional load imbalance is also caused by the BWA code not anticipating certain hardware characteristics of multi-socket multicores, such as the non-uniform memory access time of the different cores. We show that rewriting the parallel section using Cilk removes the load imbalance, resulting in a factor two performance improvement over the original BWA.

Keywords: BWA · Multithreading · NUMA · Load balancing · Cilk

1 Introduction

Burrows-Wheeler Aligner (BWA) [1] by Li and Durbin is a widely used short read alignment tool. It uses the Burrows-Wheeler transformation of the reference genome, which not only minimises the memory needed to store the reference, but also allows for a strategy for matching the reads that operates in the order of the read length. The technique was originally proposed in the context of text compression [5] and the matching process needed to be adapted for short read alignment to handle mismatches due to mutations (such as SNPs) and indels [1]. There are different options to handling mismatches, and BWA presents one solution. Other short read aligners based on Burrows-Wheeler transformation, such as Bowtie and SOAP2, use different strategies for mismatches, which are considered to produce faster albeit less accurate results than BWA [1,6,7].

In order to reduce the overall execution time, BWA supports multithreaded execution when appropriate hardware resources are available. In this mode, the reads are evenly distributed over the available cores of a multicore processor so that they can be aligned in parallel. In theory, this should give a linear speedup compared to sequential or single-core execution.

To evaluate the effectiveness of BWA's multithreaded mode, we set up a scaling experiment on both a 12-core and a 40-core multi-socket processor. The workload we use is a read set from the 1000 Genomes Project [8] (NA20589) with approximately 15 million of soft-clipped reads between 37–108 bp. Figure 1 shows the scaling of our workload on an Intel Xeon X5660 processor with 12 cores (2 sockets $\times$ 6 cores). BWA does not achieve linear speedup (red line). At 12 threads, we measure that BWA achieves 9x speedup, 73 % of the potential, linear, speedup (blue line). The scaling behaviour of BWA gets worse as the number of cores and sockets of the target processor increases. Figure 2 shows the scaling behaviour of our workload on a 40-core Intel Xeon E7-4870 processor (4 sockets $\times$ 10 cores). Again, BWA does not achieve linear speedup (red line). The speedup measured for 40 cores indicates a speedup of 13.9x, only 35 % of the potential (blue line).

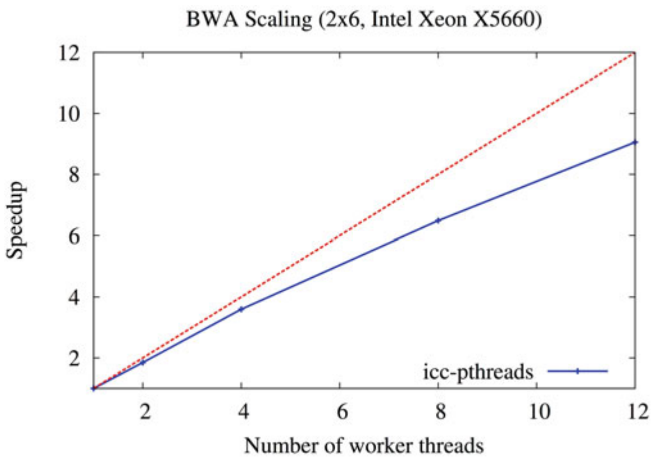


Fig. 1. Scaling of BWA on 12 cores. Red: linear speedup. Blue: measured speedup (Color figure online).

Our hypothesis is that the bad scaling behaviour of BWA is due to the fact that the parallelisation of BWA does not take into account load balancing. BWA evenly distributes the reads over the available cores at the beginning of the execution, but this may cause load imbalance when different reads require different times to process. In the worst case, an *unlucky* core gets all the *difficult* reads so that it still has to work while other cores are idle because they finished aligning their *cheap* reads. We claim that the cost of aligning reads varies because both the read length and numbers of mutations varies for the different reads in a workload. Also, the memory layout and the NUMA architecture of our multi-socket processors has an impact on the alignment cost of individual reads.

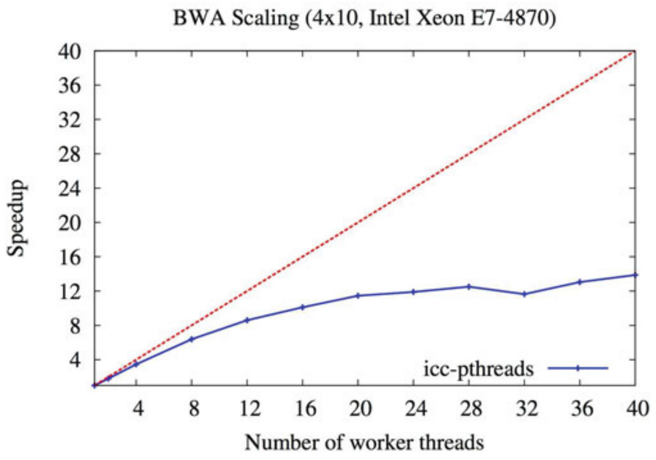


Fig. 2. Scaling of BWA on 40 cores. Red: linear speedup. Blue: measured speedup (Color figure online).

In the rest of this paper, we analyse the cause of the load imbalance in BWA in more detail. We also present a Cilk-based parallelisation strategy that removes the load imbalance, allowing us to achieve more than a factor two speedup compared to the original code.

2 BWA Implementation

Short read alignment is a massively data parallel problem. In a typical workload millions of reads, up to 100 bp (200 characters) long, need to be aligned. Reads are aligned independently from one another. Hence read alignment can be parallelised as a data parallel loop over the read set.

Concretely, the BWA code[1] sets up pthreads equal to the number of cores on the target processor.[2] Each pthread executes a sequential alignment loop for an equal share of the reads. Linear speedup for such an implementation is only guaranteed if the work to be done is roughly equal for each pthread, in order to avoid load imbalance. To detect if there is load imbalance possible, we inspect the algorithm that is executed to align reads.

2.1 Burrows-Wheeler Alignment Algorithm

The algorithm underlying the BWA code is well-documented [1], but we repeat it briefly to discuss the challenges it presents for efficient multithreaded execution.

The Burrows-Wheeler alignment algorithm relies on the definition of two auxiliary data structures. These data structures are defined in terms of a compressible version of the reference, which is created via the so-called Burrows-Wheeler

[1] We always refer to the latest version of BWA, i.e. the bwa-0.6.2 download on [2].
[2] This is actually configured via the `-t` parameter.

transformation. E.g. $BWT(abracadabra)$ would be $ard\$rcaaaabb$. Given the Burrows-Wheeler transformation of the reference, the table *c_tab* stores for each character c in the (genetic) alphabet how many characters occur in the transformation that are lexicographically smaller than c. A second table, *occ_tab* is defined so that a function $occ(occ\_tab, c, k)$ returns the number of occurrences of the character c in the prefix $BWT(ref)[1...k]$. In principle, the table *occ_tab* has for each character as many entries as the length of the reference, but BWA only stores the information for every 32 characters. For the human reference, *occ_tab* is around 3 GB large [1].

Given the tables *c_tab* and *occ_tab*, finding out where (or whether) a read matches the reference, is a *simple* calculation. Figure 3 shows pseudo code for matching a read. The code consists of a loop that iterates over the characters of the read (r). Each iteration references the *occ_tab* and *c_tab* to compute a new starting point (sp) and end point (ep), which represent a range from which the indexes—where the read matches the reference—can be calculated. The code in Fig. 3 actually only works for reads that match the reference exactly. For reads with mutations or indels, additional work is needed. For inexact matches, multiple alternative matches are checked and explored using a priority queue to direct the order in which the alternatives are explored. It is not important at this point to understand all the details, the structure of the code remains roughly the same as in Fig. 3. What is important to note is that for inexact matches additional work is necessary. This is also observed by Li et al. in their GPU implementation of SOAP2 [7].

The code in Fig. 3 embodies certain patterns that have consequences for multithreaded code:

1. The ratio of memory operations versus other operations is high: 28 % (computed with Intel® VTune™ Amplifier XE 2013). Memory operations may have a high latency and stall processors.
2. In standard multicore servers, cores are clustered in sockets. Cores on different sockets have different access times to different regions in memory, cf. non-uniform memory access (NUMA). architecture. A core can access memory on its own socket faster than memory on other, remote sockets. By default, each pthread allocates memory on its own socket. In BWA, the tables *c_tab* and *occ_tab* are allocated at the beginning of the execution, before the alignment starts. This means that all pthreads which are not on the first socket will have slower access time to these tables.
3. Aligning a read that matches some part of the reference exactly is cheaper than matching a read that has mutations or indels.
4. Reads have varying lengths when quality clipping is used—in our example workload between 37–100 bp. Since each character of a read needs to be processed by the loop in Fig. 3, longer reads will take longer to match.

The above points are all sources for load imbalance amongst the pthreads: There is load imbalance because certain threads will have slower access to the *c_tab* and *occ_tab* tables, and there is load imbalance because certain threads will have to handle longer or more mutated reads than others.

```
def exact_bwc(r):
  n = len(r)
  i = n - 1
  c = r[i]
  sp =  c_tab[c] + 1
  ep =  c_tab[next_letter(c, abc)]
  j = i - 1
  while not(j < 0 or sp > ep):
    nc = r[j]
    sp =  c_tab[nc] + occ( occ_tab, nc, sp - 1) + 1
    ep =  c_tab[nc] + occ( occ_tab, nc, ep)
    j -= 1
  return (ep - sp + 1, sp, ep)
```

Fig. 3. Alignment of a read (exact)

2.2 Measuring Load Imbalance

We can confirm the predicted load imbalance by measuring the average time each pthread needs to align its read set. Figure 4 shows the time we measure per pthread on our 12-core processor.[3] Figure 5 shows the same for the 40-core processor. In both cases, there is a clear load imbalance between the pthreads.

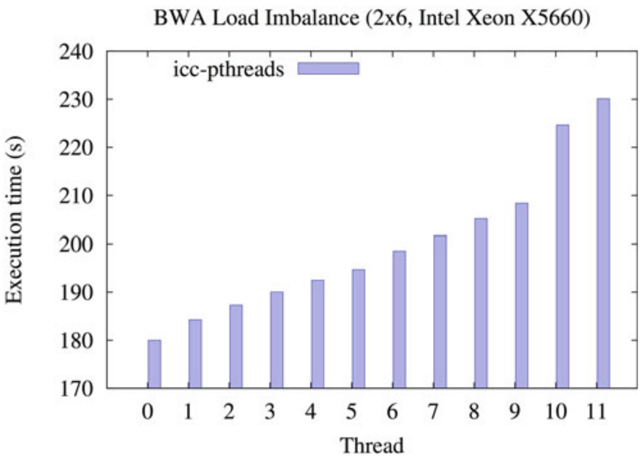


Fig. 4. BWA load imbalance on 12 cores.

3 Removing Load Imbalance with Cilk

Intel® Cilk™ Plus [3,4] is an extension for C/C++ for task-based parallel programming. It provides constructs for expressing fork/join patterns and parallel `for` loops. These constructs are mapped onto tasks that are executed by a dynamic work-stealing scheduler. With work stealing, a *worker* thread is created

[3] Averages for 5 runs. Same distribution of reads across cores for each run.

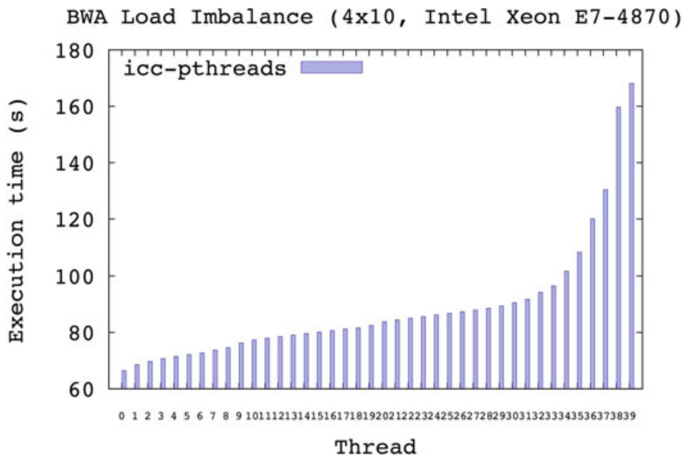


Fig. 5. BWA load imbalance on 40 cores.

for each core. Every worker thread has its own task pool, but when a worker thread runs out of tasks, it *steals* tasks from worker threads that are still busy. This way faster threads take over work from slower threads, balancing the overall workload.

The advantage of using Cilk is that load imbalance amongst threads is handled implicitly by the work-stealing scheduler. The programmer simply focuses on identifying and creating the parallelism.

3.1 Cilk-Based Parallelisation

We replace the pthread-based parallel loop in BWA by a Cilk `for` loop. There are some intricacies with regard to making sure that each worker thread has its own priority queue for intermediate matches, to avoid contention of a shared queue. Our solution is to initialise the priority queues before executing the parallel loop, one for each worker thread. The priority queues are stored in a global array so that they are globally accessible by the worker threads. Inside the `for` loop, we use Cilk's introspective operator for querying the running worker thread's ID, which we then use to identify the priority queue the worker thread accesses.

3.2 Improved Scaling Results

By using Cilk, the scaling behaviour of BWA improves drastically. Figure 6 compares the Cilk-based scaling (green) with the original pthread code (blue) on a 12-core Intel Xeon X5660 processor (2 sockets × 6 cores). To allow direct comparison, our speedup graphs use the same baseline: 1-threaded execution of unmodified BWA. With the Cilk version, we achieve a factor 10x speedup or 82 % of the potential linear speedup (red), compared to 9x speedup or 73 % for the pthread version. The results are even better for the 40-core Intel Xeon E7-4870 processor (4 sockets × 10 cores), cf. Fig. 7. There the Cilk version achieves 30x speedup, 75 % of the potential, versus 13.86x speedup or 35 % for the pthread version. The difference in improvement between the 12-core and 40-core processors is due to the fact that the 12-core processor has 2 sockets, whereas the 40-core processor has 4. Hence in case of the 40-core processor, cores are more distant from each other and load imbalance due to remote memory access is more severe. Our companion technical report [10] offers a more detailed technical discussion of these findings, as well as additional experiments with different data sets.

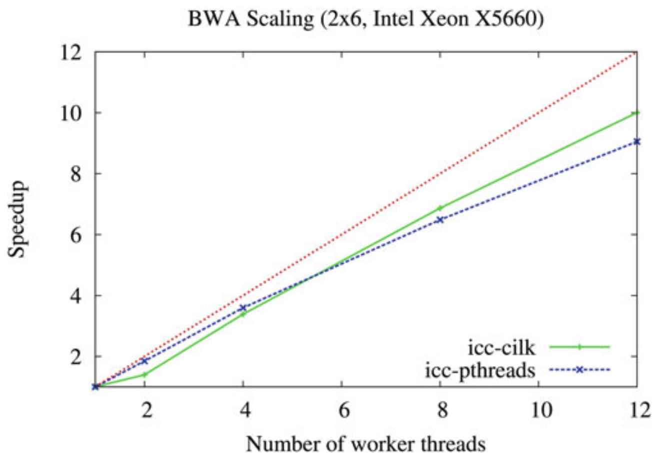


Fig. 6. Scaling of BWA on 12 cores using Cilk. Red: linear speedup. Blue: speedup measured for original pthreads implementation. Green: speedup measured for our Cilk solution (Color figure online).

4 Other Issues

Beyond load balancing, there are a number of other issues with BWA that hamper efficient multithreaded execution.

4.1 Memory Latency and Hyperthreading

When discussing the BWA algorithm in Sect. 2.1, we saw that the ratio of memory operations versus other operations is high (28 %). Memory operations have high latency and stall the processor. This is worsened by the fact that the data access pattern is random so that both caching and speculation often fail. Hyperthreading can help in multithreaded applications where the threads have bad latencies. We see a positive effect of hyperthreading with the Cilk-based version of BWA, achieving super linear speedup with regard to the number of cores. In contrast, activating hyperthreading has almost no effect for the original pthread-based BWA.[4] Further improvements using prefetching may be possible.

[4] Graphs omitted due to space restrictions, see our technical report [10].

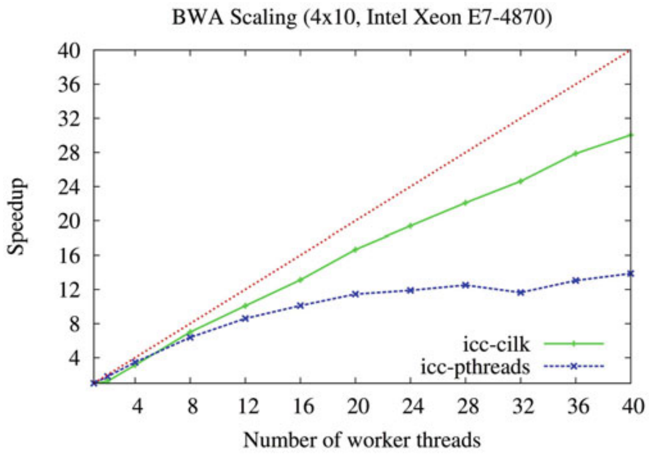


Fig. 7. Scaling of BWA on 40 cores using Cilk. Red: linear speedup. Blue: speedup measured for original pthreads implementation. Green: speedup measured for our Cilk solution (Color figure online).

4.2 Parallel Versus Sequential Section

The graphs we showed so far only take into account the time spent in the parallel section of BWA. However, the parallel section only comprises the alignment of the different reads, but before the reads can be aligned, data structures need to be initialised, e.g. loading the reads from a file into memory. This part of the code makes up the sequential section of BWA as it is not parallelised.

Amdahl's law states that the speedup to expect by parallelising a program is limited by the time spent in the sequential section. Figure 8 shows the timings for running BWA on 1 to 40 threads on our sample workload. The red part of a timing shows the time spent on sequential execution, whereas the blue part shows the time spent in the parallel section. As the number of threads increases, the time spent in the sequential section becomes a dominating factor in the overall execution time.

Using Amdahl's law, we can predict the scaling behaviour of a program. Figure 9 shows this for BWA: The red line is the ideal scaling behaviour to expect when the parallel section scales linearly, but the sequential section stays constant. The blue and green lines show the speedups we actually *measure* for both the original BWA code and our Cilk version. The black line shows linear speedup with regard to the available cores. We see that the red line is little more than half of this. If we want BWA to get closer to linear speedup to reach the potential of our processor, we need to parallelise or reduce the sequential section substantially.

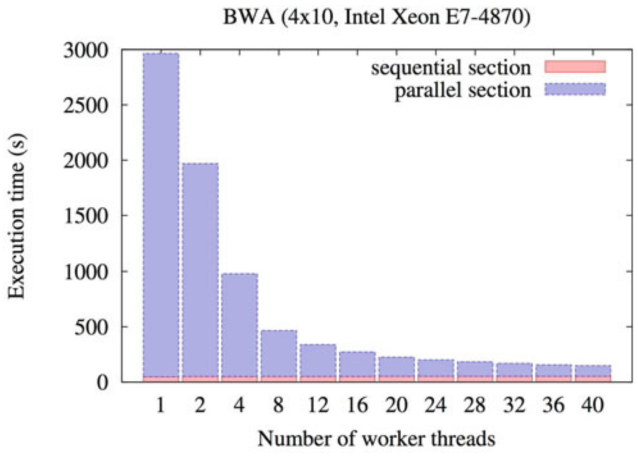


Fig. 8. Time spent on the parallel (blue) versus the sequential section (red) (Color figure online).

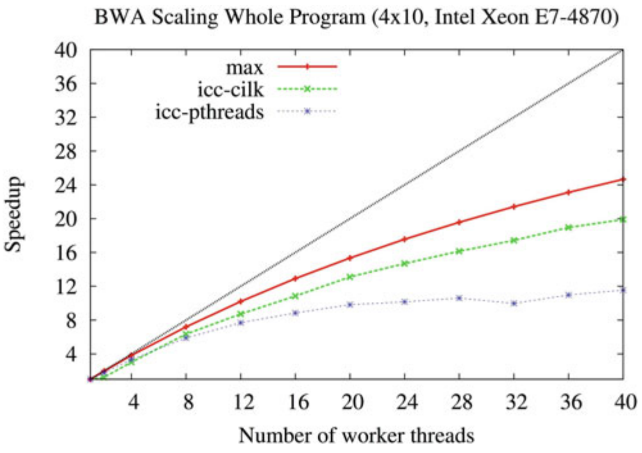


Fig. 9. BWA overall scaling versus potential scaling. Black: linear speedup. Red: theoretical speedup via Amdhal's law, when the parallel section would scale linearly. Blue: speedup measured for original BWA parallelisation using pthreads. Green: speedup measured for our Cilk solution (Color figure online).

5 Related Work

Parallel BWA (pBWA) [9] is an MPI-based implementation of BWA for cluster-based alignment, focusing on inter-node level parallelism. The improvements they claim for the multithreaded mode of BWA on a single node are already integrated with the (latest) version of BWA (bwa-0.6.2) that we adapted. Hence that work is complementary to ours.

6 Conclusions

The multithreaded mode of BWA scales poorly on multi-socket multicore processors because the parallelisation strategy, which evenly distributes the reads amongst available cores, suffers from load imbalance. We remove the load imbalance by rewriting the parallel section of BWA in Cilk, a task parallel extension for C/C++ based on a work-stealing scheduler that is capable of dynamically load balancing running programs. Using Cilk, we improve the scaling behaviour of BWA by more than a factor two, as shown by our experiments on both a 12-core and a 40-core processor. We refer the reader our technical report for experiments with additional data sets and a more detailed discussion [10].

Other issues to investigate in the future include the possible latency and bandwidth problems caused by the high number of memory operations, strategies for further reducing the NUMA penalties such as replication of data structures, as well as reducing the proportionally large sequential section.

Acknowledgments. This work is funded by Intel, Janssen Pharmaceutica and by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT).

References

1. Li, H., Durbin, R.: Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics **25**(14), 1754–1760 (2009)
2. Burrows-Wheeler Aligner. http://bio-bwa.sourceforge.net/
3. Leiserson, C.E.: The Cilk++ concurrency platform. J. Supercomput. **51**(3), 244–257 (2010). (Kluwer Academic Publishers)
4. Intel Cilk Plus. http://software.intel.com/en-us/intel-cilk-plus
5. Farragina, P., Manzini, G.: Opportunistic data structures with applications. In: 41st IEEE Annual Symposium on Foundations of Computer Science, pp. 390–398. IEEE Computer Society, Los Alamitos (2000)
6. Langmead, B., Trapnell, C., Pop, M., Salzberg, S.L.: Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biol. **10**, R25:1–R25:10 (2009). (Article: R25)
7. Li, R., Yu, C., et al.: SOAP2: an improved ultrafast tool for short read alignment. Bioinformatics **25**(15), 1966–1967 (2009)
8. Genomes Project. http://www.1000genomes.org/
9. Peters, D., Luo, X., Qiu, K., Liang, P.: Speeding up large-scale next generation sequencing data analysis with pBWA. J. Appl. Bioinform. Comput. Biol. **1**(1), 1–6 (2012)
10. Herzeel, C., Costanza, P., Ashby, T., Wuyts, R.: Performance analysis of BWA alignment. Technical report, ExaScience Life Lab (2013)

K-mulus: Strategies for BLAST in the Cloud

Christopher M. Hill[1](✉), Carl H. Albach[1], Sebastian G. Angel[2], and Mihai Pop[1]

[1] Center for Bioinformatics and Computational Biology, University of Maryland, College Park, MD, USA
{cmhill,calbach,mpop}@umd.edu
[2] University of Texas, Austin, TX, USA
sebs@cs.utexas.edu

Abstract. With the increased availability of next-generation sequencing technologies, researchers are gathering more data than they are able to process and analyze. One of the most widely performed analysis is identifying regions of similarity between DNA or protein sequences using the Basic Local Alignment Search Tool, or BLAST. Due to the large amount of sequencing data produced, parallel implementations of BLAST are needed to process the data in a timely manner. While these implementations have been designed for those researchers with access to computing grids, recent web-based services, such as Amazon's Elastic Compute Cloud, now offer scalable, pay-as-you-go computing. In this paper, we present K-mulus, an application that performs distributed BLAST queries via Hadoop MapReduce using a collection of established parallelization strategies. In addition, we provide a method to speedup BLAST by clustering the sequence database to reduce the search space for a given query. Our results show that users must take into account the size of the BLAST database and memory of the underlying hardware to efficiently carry out the BLAST queries in parallel. Finally, we show that while our database clustering and indexing approach offers a significant theoretical speedup, in practice the distribution of protein sequences prevents this potential from being realized.

Keywords: Bioinformatics · Cloud computing · Sequence alignment · Hadoop

1 Introduction

Identifying regions of similarity between DNA or protein sequences is one of the most widely studied problems in bioinformatics. These similarities can be the result of functional, structural, or evolutionary relationships between the sequences. As a result, many tools have been developed with the intention of efficiently searching for these similarities [1,5,8]. The most widely used application is the Basic Local Alignment Search Tool, or BLAST [1].

With the increased availability of next-generation sequencing technologies, researchers are gathering more data than ever before. This large influx of data has become a major issue as researchers have a difficult time processing and analyzing it. For this reason, optimizing the performance of BLAST and developing new alignment tools has been a well researched topic over the past few years. Take the example of environmental sequencing projects, in which the biodiversity of various environments, including the human microbiome, is analyzed and characterized to generate on the order of several terabytes of data [16]. One common way in which biologists use these massive quantities of data is by running BLAST on large sets of unprocessed, repetitive reads to identify putative genes [11,15]. Unfortunately, performing this task in a timely manner while dealing with terabytes of data far exceeds the capabilities of most existing BLAST implementations.

As a result of this trend, large sequencing projects require the utilization of high-performance and distributed systems. BLAST implementations have been created for popular distributed platforms such as Condor [12] and MPI [2,4]. Recently, MapReduce [3] has become one of the de-facto standards for distributed processing. There are a few advantages of using the MapReduce framework over other existing parallel processing frameworks. The entirety of the framework rests in two simple methods: a *map* and a *reduce* function. The underlying framework takes care of the communication between nodes in the cluster. By abstracting the communication between nodes, it allows software developers to quickly design software that can run in parallel over potentially thousands of processors. Although this makes it simple to program, without direct control of the communication, it may be more inefficient compared to other distributed platforms.

While these parallel implementations of BLAST were designed to work on large computing grids, most researchers do not have access to these types of clusters, due to their high cost and maintenance requirements. Fortunately, cloud computing offers a solution to this problem, allowing researchers to run their jobs on demand without the need of owning or managing any large infrastructure. Web-based services, such as Amazons Elastic Compute Cloud (EC2) [7], have risen in recent years to address the need for scalable, pay-as-you-go computing. These services allow users to select from a collection of pre-configured disk images and services, and also allow more fine-grained customization down to the number of CPUs, speed, and amount of memory in their rented cluster.

In this paper, we present K-mulus, a collection of Hadoop MapReduce tools for performing distributed BLAST queries. We show that a limitation to previous cloud BLAST implementations is their “one size fits all” solution to parallelizing BLAST queries. We provide several different strategies for parallelizing BLAST depending on the underlying cloud architecture and sequencing data, including: (1) parallelizing on the input queries, (2) parallelizing on the database, and then a (3) hybrid, query and database parallelization approach. Finally, we describe a k-mer indexing heuristic to achieve speedups by generating database clusters which results in a reduction of the search space during query execution.

2 Methods

2.1 MapReduce

The MapReduce framework was created by Google to support large-scale parallel execution of data intensive applications using commodity hardware [3]. Unlike other parallel programming framework where developers must explicitly handle inter-process communication, MapReduce developers only have to focus on two major functions, called *map* and *reduce*.

Prior to running a MapReduce program, the data must be first stored in the Hadoop Distributed File System (HDFS). The user then specifies a *map* function that will run on the chunks of the input data in parallel. MapReduce is "data aware," performing computation at the nodes containing the required data instead of transferring the data across the network. The *map* function processes the input in a particular way according to the developers specifications, and outputs a series of key-value pairs. Once all nodes have finished outputting their key-value pairs, all the values for a given key are aggregated into a list (via Hadoop's internal shuffle and sort mechanisms), and sent to the assigned reducer. During the reduce phase, the (key, list of values) pairs are processed. This list of values is used to compute the final result according to the applications needs. For more details and examples, please see [3].

2.2 Parallelization Strategies

K-mulus uses three main strategies to perform distributed BLAST queries using Hadoop MapReduce. As we will show, the efficacy of these strategies are all dependent on the underlying hardware and data being used.

Query Segmentation. Arguably the simplest way to parallelize an application using MapReduce is to set the *map* function to the given application and execute it on subsets of the input. The individual results of the *map* functions are then aggregated by a single *reducer*. This query segmentation is the default

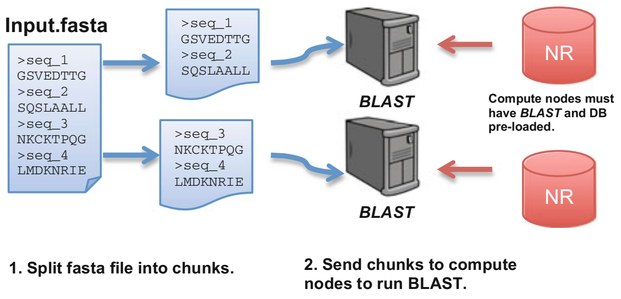


Fig. 1. Query segmentation approach for parallelizing BLAST.

implementation of CloudBLAST [13], a popular MapReduce implementation of BLAST. Instead of writing custom *map* and *reduce* functions, CloudBLAST takes advantage of Hadoop's streaming extension that allows seamless, parallel execution of existing software on Hadoop without having to modify the underlying application.

The first step of the query segmentation approach is to partition the query file into a predetermined number of chunks (usually the number of computing nodes) and send them to random nodes (Fig. 1). This partitioning of the query sequences can be done automatically as the sequence files are uploaded to the HDFS. The user must pay special attention to the size of their query sequence file because the block sizes for the HDFS are 128 MB by default. It is possible to underutilize the Hadoop cluster, since the *map* functions are often assigned to blocks of the input data. If a user uploads a 128 MB sequence file to HDFS and uses Hadoop's streaming extension, then despite the number of nodes they request, BLAST will be performed only using the node containing the block of the data.

During runtime, the *map* function receives as input a block of FASTA-formatted sequences. Each *map* function simply executes the included BLAST binary against the included database sequences and outputs the results directly to disk. Although there is no need to use a reducing step for this strategy, one reducer can be used to aggregate the results.

Database Segmentation. Instead of segmenting the query, we can segment the database into a predetermined number of chunks. By segmenting the database, we can reduce the overhead of disk I/O for databases that do not fit completely into memory. Otherwise, as soon as the database grows larger than the amount of main memory, the runtime increases by orders of magnitude [2]. Therefore, it is important to examine the underlying hardware limitations and database size before using the default query segmentation approach.

During runtime, the query sequences are uploaded to the HDFS and sent to all nodes using the DistributedCache feature of Hadoop. The DistributedCache feature ensures that all nodes involved in the MapReduce have access to the same files. The *map* function is only responsible for passing the path of the database chunks to the reducer. Each *reduce* function executes BLAST on the complete set of input sequences.

Since BLAST takes into account the size of the database when computing alignment statistics, the individual BLAST results must have their scores adjusted for the database segmentation. Fortunately, BLAST provides the user an option to specify the effective length of the complete database.

Hybrid Approach. One potential problem with the database segmentation approach is that if we evenly partition the database across all nodes in our cluster, then the database chunks may only fill up a small portion of the available memory. In this case, we must use a hybrid approach, where we segment the database into the least number of chunks that can fit entirely into memory.

Afterwards, we replicate the database chunks across the remaining machines. During runtime, the query sequences are split and sent to the different the databases chunks, but only sent once to each of the database chunk replicates. This hybrid approach is also utilized by *mpiBLAST* [2], a widely-used distributed version of BLAST using MPI, which can yield super-linear speed-up over running BLAST on a single node.

During runtime, each *map* function receives a chunk of the query sequences and is responsible for sending out the chunk to each database partition. For each database partition i, the *map* function randomly selects a replicate to send the query chunk to in the form of a tuple ($db_{i,\text{replicate\_num}}$, query chunk). The reducer receives a collection of query chunks for a given database partition and replicate and BLASTs the query chunk against the database partition.

2.3 K-mer Indexing

One of the original algorithms that BLAST uses is "seed and extend" alignment. This approach requires that there be at least one k-mer (sequence sub-string of length k) match between query and database sequence before running the expensive alignment algorithm between them [1]. Using this rule, BLAST can bypass any database sequence which does not share any common k-mers with the query. Using this heuristic, we can design a distributed version of BLAST using the MapReduce model. One aspect of BLAST which we take advantage of is the database indexing of k-mers. While some versions of BLAST have adopted database k-mer indexing for DNA databases, it seems that this approach has not been feasibly scaled to protein databases [14]. For this reason, BLAST iterates through nearly every database sequence to find k-mer hits. Here we describe an approach for K-mulus that attempts to optimize this process by using lightweight database indexing to allow query iteration to bypass certain partitions of the database.

In order to cluster the database, for each sequence, we first create a vector of bits in which the value at each position indicates the presence of a specific sequence k-mer. The index of each k-mer in the vector is trivial to compute. We then cluster these bit vectors using a collection of clustering algorithms: k-means [6], and k-medoid [9]. Our algorithms perform clustering with respect to the presence vectors of each input sequence. For each cluster, a center presence vector is computed as the union of all sequence presence vectors in the cluster. The distance between clusters is taken as the Hamming distance, or number of bitwise differences, between these cluster centers. This design choice creates a tighter correspondence between the clustering algorithm and the metrics for success of the results, which depend entirely on the cluster presence vectors as computed above. We also keep track of the centers for each cluster as they play the crucial role of identifying membership to a cluster.

After the database has been clustered, we compare the input query sequences to all centers. The key idea is that by comparing the input query sequence to the cluster centers, we can determine whether a potential match is present in a given cluster. If this is the case, we run the BLAST algorithm on the query sequence and the database clusters that we determined as relevant for the query.

3 Results

3.1 Comparison of Parallelization Approaches on a Modest Size Cluster

We evaluated the different parallelization approaches of protein BLAST on 30,000 translated protein sequences randomly chosen from the Human Microbiome Project [16] (Fig. 2). The sequences were BLAST against NCBI's non-redundant (*nr*) protein database (containing 3,429,135 sequences). For our analyses we used a 46 node Hadoop (version 0.20.2) cluster. Each node had 2 map/reduce tasks and 2 GB of memory, reproducing a typical cloud cluster.

The *nr* database used was 9 GB in size and unable to completely fit into the memory of a single node in our cluster. We segmented the database into 100 and 500 chunks to test our database segmentation approach. With 100 database chunks, the database will be roughly split across each reduce task. We included a partitioning of 500 database chunks to show the effects of over-partitioning the database.

Segmenting the database into 100 and 500 partitions resulted in a 26 % and 16 % decrease in runtime compared to the query segmentation approach, respectively. Although using a smaller number of database partitions was faster, there are still advantages for using more database partitions. Assuming an even distribution of query workload, if a node fails near the end of its BLAST execution, then that task must be restarted and the overall runtime is essentially doubled. Over-partitioning the database allows for a failed task to restart and complete faster.

Our hybrid query and database segmentation approach resulted in a 44 % decrease in runtime compared to only query segmentation. Considering that the memory of each node in our cluster was 2 GB, and the *nr* database was 9 GB, we partitioned the database into 5 chunks, each roughly 2 GB in size. This allows the databases to fit completely into memory at each node.

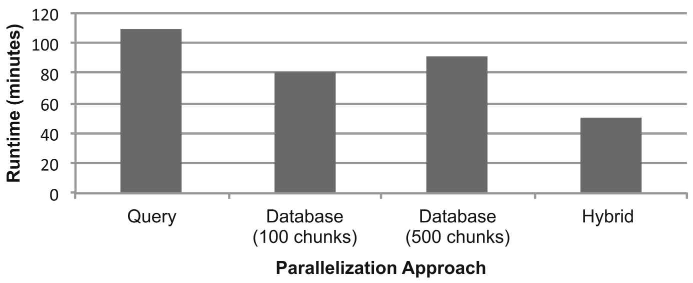


Fig. 2. Runtimes of different BLAST parallelization approaches.

3.2 Analysis of Database K-Mer Index

Using our clustering and k-mer index approach, we show noticeable speedups on well clustered data. To demonstrate this we simulated an ideal data set of 1,000 sequences, where the sequences were composed of one of two disjoint sets of 3-mers. The database sequences were clustered into two even-size clusters. The sample query was 10,000 sequences, also comprising one of two disjoint sets of 3-mers. Figure 3 shows the result of running BLAST on the query using Hadoop's streaming extension with query segmentation (the method used by CloudBLAST to execute BLAST queries) and K-mulus. K-mulus running on 2 cores with 2 databases yields a 56 % decrease in runtime over BLAST using Hadoop's streaming extension on 2 cores. In practice, this degree of separability is nearly impossible to replicate, but this model allows us to set a practical upper bound for the speedup contributed by clustering and search space reduction.

For a more practical BLAST query using the *nr* database, our database and k-mer indexing approach took 2.75x as long compared to the naive Hadoop streaming method using a realistic query of 30,000 sequences from the HMP project. The poor performance is due to the very high k-mer overlap between clusters and uneven cluster sizes. Due to the high k-mer overlap, each query sequence is being replicated and compared against nearly all clusters.

K-mulus' database clustering and k-mer indexing approach shows poor performance due entirely to noisy, overlapping clusters. In the worst case, K-mulus will map every query to every cluster and devolve to a naive parallelized BLAST on database segments, while also including some overhead due to database indexing. This is close to the behavior we observed when running our clustering and k-mer index experiments on the *nr* database. In order to describe the best possible clusters we could have generated from a database, we considered a lower limit on the exact k-mer overlap between single sequences in the *nr* database (Fig. 4). We generated this plot by taking 50 random samples of 3000 *nr* sequences each, computing the pairwise k-mer intersection between them, and plotting a histogram of the magnitude of pairwise k-mer overlap. This shows that there are very few sequences in the *nr* database which have no k-mer overlap which makes the generation of disjoint clusters impossible. Furthermore, this plot is optimistic

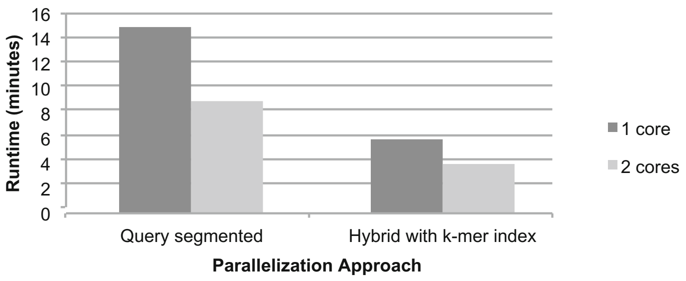


Fig. 3. Runtimes of database segmentation with k-mer index approach.

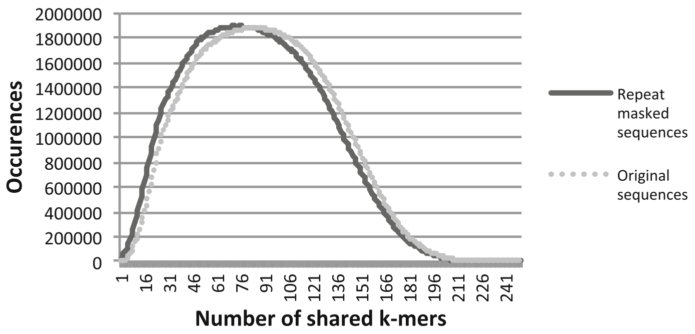


Fig. 4. Pair-wise k-mer intersection of 50 random samples of 3000 original and repeat-masked *nr* sequences.

in that it does not include BLASTs neighboring words, nor does it illustrate comparisons against cluster centers which will have intersection greater than or equal to that of a single sequence.

One strategy to improve the separability of the clusters and reduce the k-mer intersection between clusters is to use repeat masking software. In order to show the improvement offered by repeat masking, we ran SEG [17] on the sequences before computing the intersection (Fig. 4). On average, SEG resulted in a 6 % reduction in the number of exact k-mer overlap between two given sequences. Repeat masking caused a significant, favorable shift in k-mer intersection and would clearly improve clustering results. However, the *nr* database had so much existing k-mer overlap that using SEG preprocessing would have almost no effect on the speed of K-mulus' clustering and k-mer index approach.

4 Discussion

With Amazon EC2 and other cloud platforms supporting Hadoop, developers should not make assumptions about the underlying hardware. Here we have provided K-mulus, which gives users the versatility to handle the common ways to perform distributed BLAST queries in the cloud without making assumptions of the underlying hardware and data. The default approach of most Hadoop implementations of BLAST is to segment the query sequences and run BLAST on the chunks in parallel. This approach works best when the entire BLAST database can fit into memory of a single machine, but as sequencing becomes cheaper and faster, this will become less likely. Computing clusters provided by services such as EC2 often contain commodity hardware with low memory, which we have shown makes the default query segmentation approach poor in practice. The query segmentation approach works quite well on more powerful clusters that are able to load the entire database into memory. By providing users with the different parallel strategies, they are free to choose the one that is most effective with their data and hardware.

We have also provided a way to speed up BLAST queries by clustering and indexing the database using MapReduce. The speedup potential is largely dependent on the clusterability of the data. Protein sequences lie in high-dimensional non-Euclidean space, so by comparing them, we encounter the curse of dimensionality, where almost all pairs of sequences are equally far away from one another. This problem maybe slightly alleviated if we are trying to cluster multiple data sets of highly redundant sequences (multiple deep coverage whole genome sequencing projects with distinct, non-intersecting k-mer spectra). Future work includes clustering and indexing the query sequences, which may have higher redundancy than the database sequences.

Although our clustering and indexing approach was used on protein sequences, the logical next step is to include nucleotide database indexing, which has historically had more success in speeding up sequence alignment [8]. With a four character alphabet and simplified substitution rules, nucleotides are easier to work with than amino acids, and allow for much more efficient hashing by avoiding of the ambiguity inherent in amino acids.

It should be noted that the parallelization strategies presented here would also benefit other commonly used bioinformatics tools. Short read alignment tools (such as Bowtie2 [10]) can be parallelized by partitioning the reference index as well as the query sequences. More work needs to be done to determine the best parallelization strategies for these tools running on commodity clusters.

Availability. Java source code for K-mulus are located at: https://github.com/biocloud/k-mulus.

Acknowledgments. We would like to thank Mohammadreza Ghodsi for advice on clustering, Daniel Sommer for advice on Hadoop, Lee Mendelowitz for manuscript feedback, Katherine Fenstermacher for the name K-mulus, and the other members of the Pop lab for valuable discussions on all aspects of our work.

This work is supported in part by grants from the National Science Foundation, grant IIS-0844494 to MP.

References

1. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. J. Mol. Biol. **215**(3), 403–410 (1990)
2. Darling, A., Carey, L., Feng, W.c.: The design, implementation, and evaluation of mpiBLAST. In: Proceedings of ClusterWorld 2003 (2003)
3. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
4. Dongarra, J.J., Hempel, R., Hey, A.J., Walker, D.W.: A proposal for a user-level, message passing interface in a distributed memory environment. Technical report, Oak Ridge National Lab., TN (United States) (1993)
5. Eddy, S.R., et al.: A new generation of homology search tools based on probabilistic inference. Genome Inf. **23**, 205–211 (2009). (World Scientific)

6. Hartigan, J.A., Wong, M.A.: Algorithm as 136: a k-means clustering algorithm. J. Roy. Stat. Soc.: Ser. C (Appl. Stat.) **28**(1), 100–108 (1979)
7. Inc, A.: Amazon Elastic Compute Cloud (Amazon EC2). Amazon Inc. (2008). http://aws.amazon.com/ec2/pricing
8. Kent, W.J.: BLAT-the BLAST-like alignment tool. Genome Res. **12**(4), 656–664 (2002)
9. Van der Laan, M., Pollard, K., Bryan, J.: A new partitioning around medoids algorithm. J. Stat. Comput. Simul. **73**(8), 575–584 (2003)
10. Langmead, B., Salzberg, S.L.: Fast gapped-read alignment with Bowtie 2. Nat. Methods **9**(4), 357–359 (2012)
11. Li, Y., Luo, H.M., Sun, C., Song, J.Y., Sun, Y.Z., Wu, Q., Wang, N., Yao, H., Steinmetz, A., Chen, S.L.: EST analysis reveals putative genes involved in glycyrrhizin biosynthesis. BMC Genomics **11**(1), 268 (2010)
12. Litzkow, M., Livny, M., Mutka, M.: Condor - a hunter of idle workstations. In: Proceedings of the 8th International Conference of Distributed, Computing Systems, June 1988 (1988)
13. Matsunaga, A., Tsugawa, M., Fortes, J.: CloudBLAST: combining MapReduce and virtualization on distributed resources for bioinformatics applications. In: IEEE Fourth International Conference on eScience, 2008. eScience'08, pp. 222–229. IEEE (2008)
14. Morgulis, A., Coulouris, G., Raytselis, Y., Madden, T.L., Agarwala, R., Schäffer, A.A.: Database indexing for production MegaBLAST searches. Bioinformatics **24**(16), 1757–1764 (2008)
15. Murray, J., Larsen, J., Michaels, T., Schaafsma, A., Vallejos, C., Pauls, K.: Identification of putative genes in bean (phaseolus vulgaris) genomic (Bng) RFLP clones and their conversion to STSs. Genome **45**(6), 1013–1024 (2002)
16. Peterson, J., Garges, S., Giovanni, M., McInnes, P., Wang, L., Schloss, J.A., Bonazzi, V., McEwen, J.E., Wetterstrand, K.A., Deal, C., et al.: The NIH human microbiome project. Genome Res. **19**(12), 2317–2323 (2009)
17. Wootton, J.C., Federhen, S.: Statistics of local complexity in amino acid sequences and sequence databases. Comput. Chem. **17**(2), 149–163 (1993)

Faster GPU-Accelerated Smith-Waterman Algorithm with Alignment Backtracking for Short DNA Sequences

Yongchao Liu(✉) and Bertil Schmidt

Institut für Informatik, Johannes Gutenberg Universität Mainz, 55099 Mainz, Germany
{liuy,bertil.schmidt}@uni-mainz.de

Abstract. In this paper, we present a GPU-accelerated Smith-Waterman (SW) algorithm with Alignment Backtracking, called GSWAB, for short DNA sequences. This algorithm performs all-to-all pairwise alignments and retrieves optimal local alignments on CUDA-enabled GPUs. To facilitate fast alignment backtracking, we have investigated a tile-based SW implementation using the CUDA programming model. This tiled computing pattern enables us to more deeply explore the powerful compute capability of GPUs. We have evaluated the performance of GSWAB on a Kepler-based GeForce GTX Titan graphics card. The results show that GSWAB can achieve a performance of up to 56.8 GCUPS on large-scale datasets. Furthermore, our algorithm yields a speedup of up to 53.4 and 10.9 over MSA-CUDA (the first stage) and gpu-pairAlign on the same hardware configurations.

Keywords: Smith-Waterman · Sequence alignment · Alignment backtracking · CUDA · GPU

1 Introduction

To identify optimal local alignments, the SW algorithm [1,2] has been widely used due to its maximal sensitivity. This algorithm has also become a fundamental operation in many research areas, such as biological sequence database search [3,4], multiple sequence alignment [5,6] and short-read alignment [7–9]. The SW algorithm is a dynamic-programming-based approach with a linear space complexity and a quadratic time complexity. This quadratic runtime feature makes the SW algorithm computationally demanding for large-scale datasets, which has driven a substantial amount of research to parallelize it based on high-performance computing architectures ranging from loosely coupled to tightly-coupled ones, including clouds [10], clusters [10] and accelerators [11]. Among these architectures, recent research mainly focuses on the use of accelerators, including field programmable gate arrays (FPGAs), single instruction multiple data (SIMD) vector execution units on CPUs, multi-core Cell Broadband Engine (Cell/BE), and general-purpose GPUs, especially CUDA-enabled GPUs.

For FPGAs, Oliver *et al.* [12,13] and Li *et al.* [14] proposed the use of linear systolic arrays and custom instructions, respectively. For SIMD vector execution units on CPUs, most investigations focus on the alignment of a single sequence pair, which are generally based on two intra-task computational patterns. One is to make SIMD vectors parallel to minor diagonals in the alignment matrix [15], and the other parallel to the query sequence by means of a sequential [16] or striped [17] layout. In addition to intra-task parallelization, some inter-task parallelization approaches have been investigated [18,19]. Compared to intra-task parallelization, the major advantages of inter-task parallelization are the independent alignment computation in SIMD vectors and the runtime independence of scoring schemes. Theses two kinds of approaches provide a general framework for other accelerators with SIMD vectors, including Cell/BEs and general-purpose GPUs. On Cell/BEs, few parallel implementations have been proposed [20,21], all of which are designed based on the intra-task parallelization with the striped layout. For general-purpose GPUs, Liu et al. [22] developed an initial OpenGL-based implementation. As the advent of the CUDA programming model, a few implementations have been developed [23–30] using CUDA.

However, almost all GPU-based implementations only calculate optimal local alignment scores. MSA-CUDA [31] made the first attempt to retrieve optimal alignments on CUDA-enabled GPUs. This algorithm employs the Myers-Miller algorithm [32], whose major advantage is the probability of aligning very long sequences as the alignment retrieval works in linear space. gpu-pairAlign [27] proposed to directly record alignment moves while performing alignments. This approach stores alignment moves in four bitwise backtracking matrices and has a linear time complexity for alignment retrieval. Although the backtracking matrices take much GPU device memory, this approach has been shown to work well for short protein sequences. Recently, CUDAlign [29] has been extended to support alignment retrieval, but only for pairwise alignment of very long DNA sequences. In addition, some programs for all-to-all sequence comparison such as mpiBLAST [33], scalaBLAST [34] and pGraph [35] have been developed on other high-performance computing platforms.

In this paper, we present GSWAB, a $\underline{\text{G}}$PU-accelerated $\underline{\text{S}}$mith-$\underline{\text{W}}$aterman algorithm with $\underline{\text{A}}$lignment $\underline{\text{B}}$acktracking for short DNA sequences. This algorithm performs all-to-all pairwise alignments and traces back the optimal local alignments on GPUs. To facilitate fast alignment retrieval, we have investigated a tile-based SW parallelization using the CUDA programming model. This tiled computing pattern enables more deep exploration of the tremendous compute power of CUDA-enabled GPUs. Our performance evaluation on a Kepler-based GeForce GTX Titan graphics card show that GSWAB can yield a performance of up to 56.8 GCUPS on large-scale datasets. In addition, on the same hardware configurations, our algorithm is able to run up to 53.4× and 10.9× faster than MSA-CUDA (the first stage) and gpu-pairAlign, respectively.

2 Methods

2.1 The Smith-Waterman Algorithm

Given a sequence R, we define $R[i, j]$ to denote the substring that starts at position i and ending at position j, and $R[i]$ to denote the i-th symbol. For a sequence pair R_1 and R_2, the recurrence of the SW algorithm with affine gap penalties is defined as

$$\begin{aligned} H_{i,j} &= max\{H_{i-1,j-1} + sbt(R_1[i], R_2[j]), E_{i,j}, F_{i,j}, 0\} \\ E_{i,j} &= max\{E_{i-1,j} - \alpha, H_{i-1,j} - \beta\} \\ F_{i,j} &= max\{F_{i,j-1} - \alpha, H_{i,j-1} - \beta\} \end{aligned} \tag{1}$$

where $H_{i,j}$, $E_{i,j}$ and $F_{i,j}$ represent the local alignment score of two prefixes $R_1[1, i]$ and $R_2[1, j]$ with $R_1[i]$ aligned to $R_2[j]$, $R_1[i]$ aligned to a gap and $R_2[j]$ aligned to a gap, respectively. α is the gap extension penalty, β is the sum of gap open and extension penalties, and sbt is a scoring function (usually represented as a scoring matrix), which defines the matching and mismatching scores between symbols. For protein sequences, we have a set of well-established scoring matrices to use, such as the BLOSUM [36] and PAM [37] families. For DNA sequences, we usually assign fixed scores for matches and mismatches. The recurrence is initialized as $H_{i,0} = H_{0,j} = E_{0,j} = F_{i,0} = 0$ for $0 \leq i \leq |R_1|$ and $0 \leq j \leq |R_2|$. The optimal local alignment score is the maximal alignment score in the alignment matrix H and can be calculated in linear space.

To obtain optimal local alignment, one approach is to store all alignment moves in an alignment backtracking matrix. This alignment retrieval works in linear time complexity, but has a quadratic space complexity, making it not suitable to very long sequences. An alternative approach is to use the linear-space Myers-Miller algorithm [32], which targets global alignment. For local alignment, we require two additional runs of score-only SW algorithm to gain the global alignment range (corresponding to the optimal local alignment) before using the algorithm. This approach favors very long sequences, but has a quadratic time complexity for alignment retrieval.

2.2 GPU Architecture

A CUDA-enabled GPU is built around a fully configurable array of scalar processors (SPs) and further organizes the SPs into a set of multi-threaded streaming multiprocessors (SMs). The GPU architecture has evolved through three generations: Tesla [38], Fermi [39] and Kepler [40]. From generation to generation, some substantial changes in the architecture have been made, such as the SM architecture, the configurability of shared memory size, and local/global memory caching.

For an architecture, it may own varied number of SMs per GPU from product to product, but has a fixed number of SPs per SM. Tesla configures each SM to contain 8 SPs and Fermi 32 SPs. Kepler adopts a new SM architecture with 192 SPs per SM. In Tesla, all SPs per SM share a fixed-size 16 KB shared memory.

Fermi introduces a size-configurable shared memory (16 KB or 48 KB) for the first time, while Kepler further provides more flexible configurations (16 KB, 32 KB or 48 KB). As for local memory, the memory size per thread is up to 16 KB in Tesla. However, both Fermi and Kepler allow up to 512 KB local memory per thread. Tesla does not cache both local and global memory. From Fermi, a L1/L2 caching hierarchy has been introduced to cache local/global memory. The L1 cache is size-configurable and provides data caching for individual SMs, whereas the L2 cache is fixed-size and provides unified data caching for the whole device. In Fermi, the global memory caching in L1 can be disabled at compile time, but the local memory caching in L1 cannot. Different from Fermi, Kepler does not allow L1 to cache global memory any more, but reserves it only for local memory accesses such as register spills and stack data. Kepler further opens the 48 KB read-only data cache, which is only accessible by texture units in Fermi, to cache read-only global memory data.

2.3 Parallelization Using CUDA

Given a set of sequences, GSWAB performs all-to-all pairwise alignments using the SW algorithm and returns the optimal local alignments in the CIGAR format [41], a compact representation of short-read (short DNA sequence) alignments, for the sake of limited GPU device memory. In this algorithm, we have proposed a tile-based SW algorithm with alignment backtracking based on the CUDA programming model. This tiled computing pattern enables our algorithm to more deeply exploit the powerful compute capability of CUDA-enabled GPUs. In addition, GSWAB works on both Fermi-based and Kepler-based GPUs.

Tile-Based Smith-Waterman Algorithm. For a sequence pair R_1 and R_2, GSWAB allocates a single backtracking matrix of size $|R_1| \times |R_2|$ with 2 bits representing the value of each cell, since each cell depends on its left, upper and upper-left neighbors. In the backtracking matrix, each cell must hold one of the four alignment move types: move from the left cell (ML), move from the upper cell (MU), move from the upper-left cell (MUL), and a stop code. The stop code is assigned only if the corresponding cell value in matrix H is zero. In this way, we can accomplish alignment retrieval only by means of the backtracking matrix. For short DNA sequences of a few hundred nucleotides, we can afford the memory overhead of the backtracking matrix, which are allocated in local memory in order to benefit from the L1/L2 cache hierarchy. In our implementation, both the alignment and backtracking matrices are partitioned into small tiles of size 4×4. The whole SW algorithm computation is conducted by computing all tiles one-by-one. Figure 1 illustrates the tile-based processing of the alignment matrix.

This tile-based computation can significantly improve data access performance because of the following two reasons. First, all alignment moves in a single tile can be represented by a 32-bit integer because each cell takes 2 bits. This means that only a single write to the backtracking matrix is needed for one

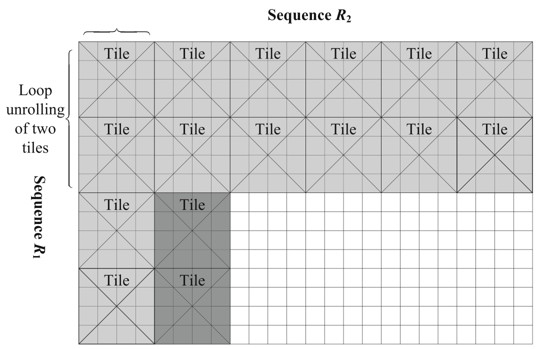


Fig. 1. Tile-based processing of the alignment matrix

tile computation, significantly reducing the number of writes to external device memory. Secondly, the data access performance to the backtracking matrix can get improved while tracing back the alignment. This is because all alignment moves in a single tile, represented as a 32-bit integer, can be realized by only a single data fetch from the backtracking matrix. While tracing back the alignment, if the next cell lies in the same tile with the current cell, we can reuse the current tile value with no need of reloading. Albeit the existence of caches, the alignment backtracking still can benefit from our data reuse, as the current tile value is possibly swapped out of the caches. Figure 2 shows the pseudo-code of our proposed approach.

Assuming R_1 and R_2 are indexed by i $(0 \leq i < |R_1|)$ and j $(0 \leq j < |R_2|)$ respectively (see Fig. 2), our alignment backtracking starts from the cell (j^*, i^*) with the maximal optimal local alignment score and does not stop until meeting either of the two cases: one is that either of the two coordinates is less than zero; and the other is that the current cell holds the stop code. During the alignment retrieval, if the value held by the current cell is ML, it means a deletion at position j^* of R_2 and subsequently we will decrease j^* by 1 and move to the next cell. If the current cell value is MU, it means a deletion at position i^* of R_1 and then we will decrease i^* by 1. If the current cell value is MLU, it means a match/mismatch and then we will decrease both i^* and j^* by 1.

As query profile is not well-suited to all-to-all pairwise alignments, we did not use it in our algorithm. Instead, we calculate the match and mismatch scores between symbols by directly comparing their values. On the other hand, the intermediate buffers for the SW algorithm are allocated in local memory, rather than global memory. This is because writable global memory accesses can only be cached by the unified L2 cache, whereas local memory can obtain additional caching from the L1 cache per SM. All reads are stored in texture memory. To facilitate the tile-based data access and to reduce the number of texture fetches, for each read we have packed four successive symbols using the integer data type with each symbol occupying 8 bits. In this manner, we can realize four symbols by a single texture fetch.

```
/*assume that the two sequence lengths are multiples of 8*/
for ($i$=0; $i$<$s1Length$; $i$+=8){      /*read $R_1$*/
    initialize related variables;
    load four consecutive symbols starting from index $i$;
    load four consecutive symbols staring from index $i + 4$;
    for ($j$=0; $j$<$s2Length$; $j$+=4){      /*Read $R_2$*/
        load the packed 4 symbols with indices [$j$, $j + 3$] to a register variable $rB$.
        /*the coordinate of tile $T_1$ is ($j$/4, $i$/4) and of tile $T_2$ is ($j$/4, $i$/4+1)*/
        resetting register variables $rA_1$ and $rA_2$ storing alignment moves for tiles $T_1$ and $T_2$.
        for ($k = 0$; $k < 4$; $k$++){
            get the ($j + k$)-th symbol of read $R_2$ from $rB$.
            load the alignment scores of cell ($j$+$k$, $i$-1) from the buffer in local memory.
            compute the $k$-th column of $T_1$, and save the alignment moves in $rA_1$ as well as
            other data such as the cell coordinate ($j^*$, $i^*$) with the maximum score.
            compute the $k$-th column of $T_2$, and save the alignment moves in $rA_2$ as well as
            other data such as the cell coordinate ($j^*$, $i^*$) with the maximum score.
            save the alignment scores of cell ($j$+$k$, $i$+7) to the buffer in local memory.
        }
        save $rA_1$ to the alignment backtracking matrix in local memory.
        save $rA_2$ to the alignment backtracking matrix in local memory.
    }
}
```

Fig. 2. Pseudocode of our tile-based SW algorithm with alignment backtracking

Alignment Launching. As the tasks of all-to-all pairwise alignments can be conceptualized as a task matrix, GSWAB distributes all alignment tasks onto the GPU using the cell-block-based task assignment approach in [31]. This task distribution divides the upper-triangle (or lower-triangle) of the whole task matrix into many equally-sized cell blocks (i.e. sub-matrices), and assigns a single thread block to process a single cell block. Within a thread block, a single thread is assigned to align a single sequence pair for simplicity. To alleviate global memory pressure on the storage of optimal alignments, we have conducted all alignments in a multi-pass way. In each pass, we calculate the number N_{TB} of thread blocks at runtime as

$$N_{TB} = \frac{C \times N_{SM} \times N_{MRT}}{N_{TPB}} \tag{2}$$

where C is a scaling factor (default = 64), N_{SM} is the number of SMs on the GPU, N_{MRT} is the maximum number of resident threads per SM, and N_{TPB} is the number of thread per thread block configured by users (default = 64).

3 Performance Evaluation

We have evaluated the performance of GSWAB using three Illumina-like single-end short-read datasets. All datasets contain 2,000 short-reads, but have different read lengths: 100, 250 and 500. To ensure that all reads in any dataset are closely related, we have generated each dataset using the Mason simulator (http://www.seqan.de/projects/mason) from a small reference sequence of 1,000 nucleotides. To measure the speed of a SW implementation, we usually use the runtime and GCUPS (billion cell updates per second) metrics. However, it should be stressed that because of the variable number of cell accesses (usually depending on sequence similarities) in the alignment retrieval procedure, the GCUPS metric might not be able to reflect the speed as precisely as the score-only cases. Nevertheless, this metric does provide a more convenient approach to facilitating users to estimate the runtime for a specific dataset. Hence, we have used the two aforementioned metrics in this paper.

All tests are conducted on a Kepler-based GeForce GTX Titan graphics card, with 14 SMs comprising 2,688 SPs and 6 GB RAM, which is installed in a personal computer with an Intel i7 2700K quad-core 3.5 GHz CPU, 16 GB memory and the Linux operating system (Ubuntu 12.04). This graphics card has a core frequency of 876 MHz, a memory clock frequency of 3,004 MHz and a L2 cache size of 1.5 MB.

We have first evaluated the performance of GSWAB using the aforementioned three datasets (see Table 1). From the table, GSWAB yields an average performance of 49.6 GCUPS, with a maximum performance of 56.8 GCUPS and a minimum of 40.0 GCUPS, for all datasets. Moreover, the speed gets improved as the read length becomes larger.

Table 1. Runtime and GCUPS of our algorithm

Dataset	#Seqs	Time (s)	GCUPS
100-bp	2,000	0.5	40.0
250-bp	2,000	2.4	52.1
500-bp	2,000	8.8	56.8

Finally, we have compared GSWAB to MSA-CUDA [42] and gpu-pairAlign [27], both of which perform all-to-all pairwise alignments. In this evaluation, for MSA-CUDA, we have only considered its first stage, which computes a pairwise distance matrix by means of all-to-all local alignments. Figure 3 shows the speedups of GSWAB over MSA-CUDA and gpu-pairAlign. Compared to MSA-CUDA, GSWAB achieves a speedup of 14.2, 23.0 and 53.4 for the 100-bp, 250-bp and 500-bp datasets, respectively. Moreover, the speedups significantly get larger as the increase of read length. Compared to gpu-pairAlign, GSWAB yields a speedup of 8.8, 10.2 and 10.9 for the 100-bp, 250-bp and 500-bp datasets,

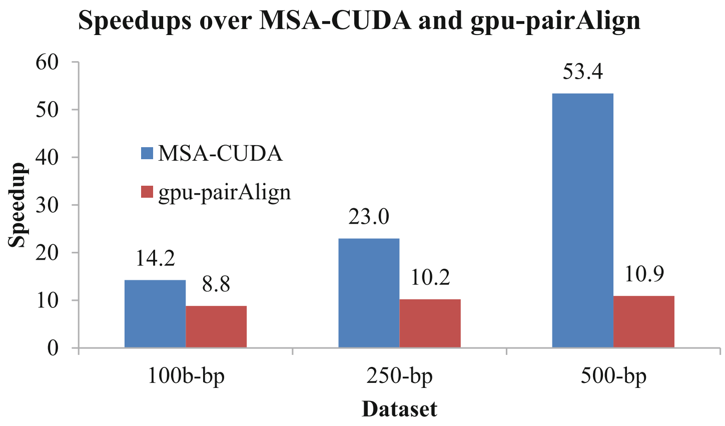


Fig. 3. Speedups over MSA-CUDA and gpu-pairAlign

respectively. Unlike MSA-CUDA, the speedups keep relatively consistent for different read lengths. This might be due to the different time complexities of the alignment backtracking, where the time complexity is quadratic for MSA-CUDA, but linear for both GSWAB and gpu-pairAlign.

4 Conclusions

We have presented GSWAB, a GPU-accelerated SW algorithm for short DNA sequences, which performs all-to-all pairwise alignments and retrieves optimal local alignments on CUDA-enabled GPUs. In GSWAB, we have investigated a tile-based SW implementation to facilitate fast alignment backtracking on GPUs. Our algorithm works on both the Fermi and Kepler architectures. The most memory-consuming data structure is the backtracking matrix, whose device memory footprint scales quadratically with the maximum allowable DNA sequence length (default = 640 and configurable at compile time). The performance of our algorithm has been evaluated on a Kepler-based GeForce GTX Titan graphics card using three simulated datasets. The performance evaluation reveals that our algorithm can produce a performance of up to 56.8 GCUPS. In addition, GSWAB is able to run up to 53.4× and 10.9× faster than MSA-CUDA (the first stage) and gpu-pairAlign on the same hardware configurations, respectively. Although our algorithm is designed for all-to-all pairwise alignments, it can be easily adopted to pure pairwise alignments with no changes of the core code. Finally, we expect that our proposed parallelization can be further used to accelerate some short-read alignment algorithms (e.g. [43–45]) based on the seed-and-extend heuristic.

References

1. Smith, T., Waterman, M.: Identification of common molecular subsequences. J. Mol. Biol. **147**, 195–197 (1981)
2. Gotoh, O.: An improved algorithm for matching biological sequences. J. Mol. Biol. **162**, 707–708 (1982)
3. Pearson, W.R., Lipman, D.J.: Improved tools for biological sequence comparison. Proc. Nat. Acad. Sci. USA **85**, 2444–2448 (1988)
4. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. J. Mol. Biol. **215**, 403–410 (1990)
5. Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence Weighting, position-specific gap penalties and weight matrix choice. Nucleic Acid Res. **22**, 4673–4680 (1994)
6. Liu, Y., Schmidt, B., Maskell, D.L.: MSAProbs: multiple sequence alignment based on pair hidden Markov models and partition function posterior probabilities. Bioinformatics **26**, 1958–1964 (2010)
7. Liu, Y., Schmidt, B., Maskell, D.L.: CUSHAW: a CUDA compatible short read aligner to large genomes based on the Burrows-Wheeler transform. Bioinformatics **28**, 1830–1837 (2012)
8. Alachiotis, N., Berger, S.A., Stamatakis, A.: Coupling SIMD and SIMT architectures to boost performance of a phylogeny-aware alignment kernel. BMC Bioinform. **13**, 196 (2012)
9. Liu, C.M., Wong, T., Wu, E., Luo, R., Yiu, S.M., Li, Y., Wang, B., Yu, C., Chu, X., Zhao, K., Li, R., Lam, T.W.: SOAP3: ultra-fast GPU-based parallel alignment tool for short reads. Bioinformatics **28**, 878–879 (2011)
10. Qiu, J., Ekanayake, J., Gunarathne, T., Choi, J.Y., Bae, S.H., Li, H., Zhang, B., Wu, T.L., Ruan, Y., Ekanayake, S., Hughes, A., Fox, G.: Hybrid cloud and cluster computing paradigms for life science applications. BMC Bioinform. **11**, S3 (2010)
11. Liu, Y., Maskell, D.L., Schmidt, B.: CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. BMC Res. Notes **2**, 73 (2009)
12. Oliver, T., Schmidt, B., Nathan, D., Clemens, R., Maskell, D.L.: Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW. Bioinformatics **21**, 3431–3432 (2005)
13. Oliver, T., Schmidt, B., Maskell, D.L.: Reconfigurable architectures for bio-sequence database scanning on FPGAs. IEEE Trans. Circuit Syst. II **52**, 851–855 (2005)
14. Li, T.I., Shum, W., Truong, K.: 160-fold acceleration of the Smith-Waterman algorithm using a Field Programmable Gate Array (FPGA). BMC Bioinform. **8**, I85 (2007)
15. Wozniak, A.: Using video-oriented instructions to speed up sequence comparison. Comput. Appl. Biosci. **13**, 145–150 (1997)
16. Rognes, T., Seeberg, E.: Six-fold speedup of Smith-Waterman sequence database searches using parallel processing on common microprocessors. Bioinformatics **16**, 699–706 (2000)
17. Farrar, M.: Striped Smith-Waterman speeds database searches six times over other SIMD implementations. Bioinformatics **23**, 156–161 (2007)
18. Alpern, B., Carter, L., Gatlin, K.S.: Microparallelism and high performance protein matching. In: Proceedings of the 1995 ACM/IEEE Supercomputing Conference (1995)

19. Rognes, T.: Faster Smith-Waterman database searches with inter-sequence SIMD parallelization. BMC Bioinform. **12**, 221 (2011)
20. Wirawan, A., Kwoh, C.K., Hieu, N.T., Schmidt, B.: CBESW: sequence alignment on Playstation 3. BMC Bioinform. **9**, 377 (2008)
21. Szalkowski, A., Ledergerber, C., Krahenbuhl, P., Dessimoz, C.: SWPS3 fast multi-threaded vectorized Smith-Waterman for IBM Cell/B.E. and x86/SSE2. BMC Res. Notes **1**, 107 (2008)
22. Liu, W., Schmidt, B., Voss, G., Muller-Wittig, W.: Streaming algorithms for biological sequence alignment on GPUs. IEEE Trans. Parallel Distrib. Syst. **18**, 1270–1281 (2007)
23. Manavski, S.A., Valle, G.: CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. BMC Bioinform. **9**, S10 (2008)
24. Ligowski, L., Rudnicki, W.: An efficient implementation of Smith Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases. In: 2009 IEEE International Symposium on Parallel and Distributed Processing, pp. 1–8 (2009)
25. Liu, Y., Schmidt, B., Maskel, D.L.: CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. BMC Res. Notes **3**, 93 (2010)
26. Khajeh-Saeed, A., Poole, S., Perot, J.: Acceleration of the Smith Waterman algorithm using single and multiple graphics processors. J. Comput. Phys. **229**, 4247–4258 (2010)
27. Blazewicz, J., Frohmberg, W., Kierzynka, M., Pesch, E., Wojciechowski, P.: Protein alignment algorithms with an efficient backtracking routine on multiple GPUs. BMC Bioinform. **12**, 181 (2011)
28. Hains, D., Cashero, Z., Ottenberg, M., Bohm, W., Rajopadhye, S.: Improving CUDASW++, a parallelization of Smith-Waterman for CUDA enabled devices. In: 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, pp. 490–501 (2011)
29. de Oliveira Sandes, E.F., de Melo, A.C.M.: Retrieving Smith-Waterman alignments with optimizations for megabase biological sequences using GPU. IEEE Trans. Parallel Distrib. Syst. **24**(5), 1009–1021 (2013)
30. Liu, Y., Wirawan, A., Schmidt, B.: CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions. BMC Bioinform. **14**, 117 (2013)
31. Liu, Y., Schmidt, B., Maskell, D.L.: MSA-CUDA: multiple sequence alignment on graphics processing units with CUDA. In: 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors (2009)
32. Myers, E.W., Miller, W.: Optimal alignments in linear space. Comput. Appl. Biosci. **4**, 11–17 (1988)
33. Darling, A., Carey, L., Feng, W.: The design, implementation, and evaluation of mpiBLAST. In: 4th International Conference on Linux Clusters: The HPC Revolution 2003 in Conjunction with ClusterWorld Conference and Expo (2003)
34. Oehmen, C.S., Baxter, J.: ScalaBLAST 2.0: rapid and robust BLAST calculations on multiprocessor systems. Bioinformatics **29**, 797–798 (2013)
35. Wu, C., Kalyanaraman, A., Cannon, W.R.: pGraph: efficient parallel construction of large-scale protein sequence homology graphs. IEEE Trans. Parallel Distrib. Syst. **23**, 1923–1933 (2012)
36. Henikoff, S., Henikoff, J.: Amino acid substitution matrices from protein blocks. PNAS **89**, 10915–10919 (1992)

37. Dayhoff, M., Schwartz, R., Orcutt, B.: A model of evolutionary change in proteins. In: Dayhoff, M.O. (ed.) Atlas of Protein Sequence and Structure, vol. 5, pp. 345–358. National Biomedical Research Foundation, Washington DC (1978)
38. Lindholm, E., Nickolls, J., Oberman, S., Montrym, J.: NVIDIA Tesla: a unified graphics and computing architecture. IEEE Micro **28**, 3955 (2008)
39. NVIDIA: NVIDIAs Next Generation CUDA Compute Architecture: Fermi. NVIDIA Corporation Whitepaper (2009)
40. NVIDIA: NVIDIAs Next Generation CUDA Compute Architecture: Kepler GK110. NVIDIA Corporation Whitepaper (2012)
41. Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R.: 1000 genome project data processing subgroup: the sequence alignment/map format and SAMtools. Bioinformatics **25**, 2078–2079 (2009)
42. Liu, Y., Schmidt, B., Maskell, D.L.: Parallel reconstruction of neighbor-joining trees for large multiple sequence alignments using CUDA. In: IEEE International Symposium on Parallel and Distributed Processing (2009)
43. Rizk, G., Lavenier, D.: GASSST: global alignment short sequence search tool. Bioinformatics **26**, 2534–2540 (2010)
44. Liu, Y., Schmidt, B.: Long read alignment based on maximal exact match seeds. Bioinformatics **28**, i318–i324 (2012)
45. Langmead, B., Salzberg, S.: Fast gapped-read alignment with Bowtie 2. Nat. Methods **9**, 357–359 (2012)

Accelerating String Matching on MIC Architecture for Motif Extraction

Solon P. Pissis[1,2(✉)], Christian Goll[2], Pavlos Pavlidis[3], and Alexandros Stamatakis[2]

[1] Florida Museum of Natural History, University of Florida, Gainesville, USA
solon.pissis@kcl.ac.uk
[2] Heidelberg Institute for Theoretical Studies, Heidelberg, Germany
[3] Foundation for Research and Technology – Hellas, Iraklio, Greece

Abstract. Identifying repeated factors that occur in a string of letters or common factors that occur in a set of strings represents an important task in computer science and biology. Such patterns are called *motifs*, and the process of identifying them is called *motif extraction*. In biology, motifs may correspond to functional elements in DNA, RNA, or protein molecules. In this article, we orchestrate `MoTeX`, a high-performance computing tool for MoTif eXtraction from large-scale datasets, on Many Integrated Core (MIC) architecture. `MoTeX` uses state-of-the-art algorithms for solving the fixed-length approximate string-matching problem. It comes in three flavors: a standard CPU version; an OpenMP version; and an MPI version. We compare the performance of our MIC implementation to the corresponding CPU version of `MoTeX`. Our MIC implementation accelerates the computations by a factor of ten compared to the CPU version. We also compare the performance of our MIC implementation to the corresponding OpenMP version of `MoTeX` running on modern Multicore architectures. Our MIC implementation accelerates the computations by a factor of two compared to the OpenMP version.

Keywords: Motif extraction · HPC · MIC architecture

1 Introduction

Identifying repeated factors that occur in a string of letters or common factors that occur in a set of strings represents an important task in computer science and biology. Such patterns are called *motifs*, and the process of identifying them is called *motif extraction*. Motif extraction has numerous direct applications in areas that require some form of *text mining*, that is, the process of deriving reliable information from text [5]. Here we focus on its application to molecular biology.

In biological applications, motifs correspond to functional and/or conserved DNA, RNA, or protein sequences. Alternatively, they may correspond to (recently, in evolutionary terms) duplicated genomic regions, such as transposable elements or even whole genes. It is mandatory to allow for a certain number of mismatches between different occurrences of the same motif since both, single nucleotide polymorphisms, as well as errors introduced by wet-lab sequencing platforms might have occurred. Hence, molecules that encode the same or related functions do not necessarily have *exactly* identical sequences.

A DNA motif is defined as a sequence of nucleic acids that has a specific biological function (e.g., a DNA binding site for a regulatory protein). The pattern can be fairly short, 5 to 20 base-pairs (bp) long, and is known to occur in different genes [7], or several times within the same gene [9]. The DNA motif extraction problem is the task of detecting overrepresented motifs as well as conserved motifs in a set of orthologous DNA sequences. Such conserved motifs may, for instance, be potential candidates for transcription factor binding sites.

A single *motif* is a string of letters (word) on an alphabet Σ. Given an integer error threshold e, a motif on Σ is said to *e-occur* in a string s on Σ, if the motif and a factor (substring) of y differ by a distance of e. In accordance with the pioneering work of Sagot [10], we formally define the common motifs problem as follows:

The *common motifs* problem takes as input a set $s_1, \ldots, s_N$ of strings on Σ, where $N \geq 2$, the quorum $1 \leq q \leq N$, the maximal allowed distance (error threshold) e, and the length k for the motifs. It consists in determining all motifs of length k, such that each motif e-occurs in at least q input strings. Such motifs are called *valid*. The values for k, e, and q are determined empirically.

In accordance with [3], motif extraction algorithms can be divided into two major classes: (1) *word-based* (string-based) methods that mostly rely on exhaustive enumeration, that is, counting and comparing oligonucleotide sequence (k-mer) frequencies; and (2) *probabilistic sequence* models, where the model parameters are estimated using maximum-likelihood or Bayesian inference methods.

Here, we focus on word-based methods for motif extraction. A plethora of word-based tools for motif extraction, such as RISO [6,10], YMF [11], Weeder [7], and RISOTTO [1], have already been released. The comprehensive study by Tompa *et al.* [12] compared thirteen different word-based and probabilistic methods on real and synthetic datasets, and identified Weeder and YMF—which are both word-based—as the most effective methods for motif extraction.

Very recently, we have introduced `MoTeX`, the first word-based HPC tool for MoTif eXtraction from large-scale datasets [8]. It can be used to tackle the common motifs problem by making a stricter assumption on motif validity, which we will elaborate later on. `MoTeX` is based on string algorithms for solving the so-called fixed-length approximate string-matching problem under the edit distance model [4] and under the Hamming distance model [2]. Given that $k \leq w$, where w is the size of the computer word (in practice, $w = 64$ or $w = 128$), the time complexity of this approach is $\mathcal{O}(N^2n^2)$ for the common motifs problem, where n is the average sequence length. The analogous parallel time complexity is $\mathcal{O}(N^2n^2/p)$, where p is the number of available processors.

Hence, `MoTeX` exhibits the following advantages: under the realistic assumption that $k \leq w$, time complexity does not depend on (i) the length k for the motifs (ii) the size $|\Sigma|$ of the alphabet, or (iii) the maximal allowed distance e. Given the stricter assumption on motif validity, it is guaranteed to find globally optimal solutions. Furthermore, the size of the output is linear with respect to the size of the input. In addition, MoTeX can identify motifs either under the edit distance model or the Hamming distance model. Finally, apart from the standard CPU version, MoTeX comes in two HPC flavors: the OpenMP-based version that supports the *symmetric multiprocessing programming* (SMP) paradigm; and the MPI-based version that supports the *message-passing programming* (MPP) paradigm.

In [8], we demonstrated that MoTeX can alleviate the shortcomings of current state-of-the-art tools for motif extraction from large-scale datasets. We showed how the quadratic time complexity of MoTeX can be slashed, in theory and in practice, by using parallel computations. The extensive experimental results presented in [8] are promising, both in terms of accuracy under statistical measures of significance as well as efficiency; a fact that suggests that further research and development of MoTeX is desirable. For instance, the MPI version of MoTeX requires about one hour to process the full upstream *Homo sapiens* genes dataset using 1056 processors, for *any* value of k and e, while current sequential programmes require more than two months for this task.

Our contribution. Many Integrated Core (MIC) architecture combines many cores onto a single chip, the *coprocessor*. One can write parallel programs, using the SMP paradigm, that can *offload* sections of code to run on the coprocessor—or alternatively, that run natively on the coprocessor. The compiler provides the language extensions to facilitate programming for MIC architecture such as *pragmas* to control the data transfer between the *host* CPU and the coprocessor. In this article, we orchestrate MoTeX on MIC architecture. We compare the performance of our MIC implementation, running on a single chip of MIC architecture, to the corresponding CPU version of MoTeX running on the host CPU. Our MIC implementation, using the full single-chip potential, accelerates the computations by a factor of ten compared to the CPU version. We also compare the performance of our MIC implementation to the corresponding OpenMP version of MoTeX running on a single chip of modern Multicore architectures. Our MIC implementation accelerates the computations by a factor of two compared to the OpenMP version, both using the full single-chip potential.

2 Definitions and Notation

In this section, in order to provide an overview of the algorithms used later on, we give a few definitions.

An *alphabet* Σ is a finite non-empty set whose elements are called *letters*. A *string* on an alphabet Σ is a finite, possibly empty, sequence of elements of Σ. The zero-letter sequence is called the *empty string*, and is denoted by ε. The *length* of a string x is defined as the length of the sequence associated with the string x, and is denoted by $|x|$. We denote by $x[i]$, for all $1 \leq i \leq |x|$, the letter at index i of x. Each index i, for all $1 \leq i \leq |x|$, is a position in x when $x \neq \varepsilon$. It follows that the ith letter of x is the letter at position i in x, and that $x = x[1 \mathinner{..} |x|]$.

A string x is a *factor* of a string y if there exist two strings u and v, such that $y = uxv$. Let the strings x, y, u, and v, such that $y = uxv$. If $u = \varepsilon$, then x is a *prefix* of y. If $v = \varepsilon$, then x is a *suffix* of y.

Let x be a non-empty string and y be a string. We say that there exists an (exact) *occurrence* of x in y, or, more simply, that x *occurs* (exactly) in y, when x is a factor of y. Every occurrence of x can be characterised by a position in y. Thus we say that x occurs at the *starting position* i in y when $y[i\mathinner{.\,.}i+|x|-1] = x$. It is sometimes more suitable to consider the *ending position* $i + |x| - 1$.

The *edit distance*, denoted by $\delta_E(x, y)$, for two strings x and y is defined as the minimum total cost of operations required to transform string x into string y. For simplicity, we only count the number of edit operations and consider that the cost of each edit operation is 1. The allowed operations are the following:

- `ins`: insert a letter in y, not present in x; $(\varepsilon, b),\ b \neq \varepsilon$;
- `del`: delete a letter in y, present in x; $(a, \varepsilon),\ a \neq \varepsilon$;
- `sub`: substitute a letter in y with a letter in x; $(a, b),\ a \neq b,\ a, b \neq \varepsilon$.

The *Hamming distance* δ_H is only defined on strings of the same length. For two strings x and y, $\delta_H(x, y)$ is the number of positions in which the two strings differ, that is, have different letters.

3 Algorithms

In this section, we first formally define the *fixed-length approximate string-matching* problem under the edit distance model and under the Hamming distance model. We then provide a brief description and analysis of the sequential algorithms to solve it. Finally, we show how the common motifs problem can be reduced to the fixed-length approximate string-matching problem, by using a stricter assumption than the one in the initial problem definition for the validity of motifs.

Problem 1 (Edit distance). Given a string x of length m, a string y of length n, an integer k, and an integer $e < k$, find all factors of y, which are at an edit distance less than, or equal to, e from every factor of fixed length k of x.

Problem 2 (Hamming distance). Given a string x of length m, a string y of length n, an integer k, and an integer $e < k$, find all the factors of y, which are at a Hamming distance less than, or equal to, e from every factor of fixed length k of x.

Let $\mathsf{D}[0\mathinner{.\,.}n, 0\mathinner{.\,.}m]$ be a dynamic programming (DP) matrix, where $\mathsf{D}[i, j]$ contains the edit distance between some factor $y[i'\mathinner{.\,.}i]$ of y, for some $1 \leq i' \leq i$, and factor $x[\max\{1, j - k + 1\}\mathinner{.\,.}j]$ of x, for all $1 \leq i \leq n$, $1 \leq j \leq m$. This matrix can be obtained through a straightforward $\mathcal{O}(kmn)$-time algorithm by constructing DP matrices $\mathsf{D}^s[0\mathinner{.\,.}n, 0\mathinner{.\,.}k]$, for all $1 \leq s \leq m - k + 1$, where $\mathsf{D}^s[i, j]$ is the edit distance between some factor of y ending at $y[i]$ and the prefix of length j of $x[s\mathinner{.\,.}s + k - 1]$. We obtain D by collating D^1 and the last row of D^s, for all $2 \leq s \leq m - k + 1$. We say that $x[\max\{1, j - k + 1\}\mathinner{.\,.}j]$ e-occurs in y ending at $y[i]$ *iff* $\mathsf{D}[i, j] \leq e$, for all $1 \leq j \leq m$, $1 \leq i \leq n$.

Table 1. Matrix D and matrix M

		0	1	2	3	4	5	6	7
		ϵ	G	G	G	T	C	T	A
0	ϵ	0	1	2	3	3	3	3	3
1	G	0	0	1	2	2	2	3	3
2	G	0	0	0	1	1	2	3	3
3	G	0	0	0	0	1	2	3	3
4	T	0	1	1	1	0	1	2	2
5	C	0	1	2	2	1	0	1	2
6	T	0	1	2	3	2	1	0	1
7	A	0	1	2	3	3	2	1	0

(a) Matrix D for $x := y :=$ GGGTCTA and $k := 3$

		0	1	2	3	4	5	6	7
		ϵ	G	T	G	A	A	C	T
0	ϵ	0	1	2	3	3	3	3	3
1	G	0	0	2	2	3	3	3	3
2	T	0	1	0	3	2	3	3	2
3	C	0	1	2	1	3	2	2	3
4	A	0	1	2	3	1	2	3	2
5	C	0	1	2	3	3	2	1	3
6	G	0	0	2	2	3	3	2	1
7	T	0	1	0	3	2	3	3	2

(b) Matrix M for $x :=$ GTCACGT, $y :=$ GTGAACT, and $k := 3$

Example 1. Let the string $x :=$ GGGTCTA, the string $y := x$, and $k := 3$. Table 1a illustrates matrix D. Consider, for instance, the case where $j = 6$. Column 6 contains all e-occurrences of factor $x[4\,..\,6] =$ TCT, that is the factor of length $k = 3$ ending at position 6 of x, in y. Cell $\mathsf{D}[4, 6] = 2$, tells us that there exists some factor $y[i'\,..\,4]$ of y, such that, for $i' = 2$, $\delta_E(y[2\,..\,4], x[4\,..\,6]) = 2$.

Iliopoulos, Mouchard, and Pinzon devised MaxShift [4], an algorithm with time complexity $\mathcal{O}(m\lceil k/w \rceil n)$, where w is the size of the computer word. By using word-level parallelism MaxShift can compute matrix D efficiently. The algorithm requires constant time for computing each cell $\mathsf{D}[i, j]$ by using word-level operations, assuming that $k \leq w$. In the general case it requires $\mathcal{O}(\lceil k/w \rceil)$ time. Hence, algorithm MaxShift requires time $\mathcal{O}(mn)$, under the assumption that $k \leq w$. The space complexity is $\mathcal{O}(m)$ since each row of D only depends on the immediately preceding row.

Theorem 1 ([4]). *Given a string x of length m, a string y of length n, an integer k, and the size of the computer word w, matrix D can be computed in time $\mathcal{O}(m\lceil k/w \rceil n)$.*

Let $\mathsf{M}[0\,..\,n, 0\,..\,m]$ be a DP matrix, where $\mathsf{M}[i, j]$ contains the Hamming distance between factor $y[\max\{1, i-k+1\}\,..\,i]$ of y and factor $x[\max\{1, j-k+1\}\,..\,j]$ of x, for all $1 \leq i \leq n$, $1 \leq j \leq m$. Crochemore, Iliopoulos, and Pissis devised an analogous algorithm [2] that solves the analogous problem under the Hamming distance model with the same time and space complexity.

Theorem 2 ([2]). *Given a string x of length m, a string y of length n, an integer k, and the size of the computer word w, matrix M can be computed in time $\mathcal{O}(m\lceil k/w \rceil n)$.*

Example 2. Let the string $x :=$ GTGAACT, the string $y :=$ GTCACGT, and $k := 3$. Table 1b illustrates matrix M. Consider, for instance, the case where $j = 7$.

Column 7 contains all e-occurrences of factor $x[5\,..\,7] = \texttt{ACT}$, that is, the factor of length $k = 3$ ending at position 7 of x, in y. Cell $\mathsf{M}[6, 7] = 1$, tells us that $\delta_H(y[4\,..\,6], x[5\,..\,7]) = 1$.

By making the following stricter assumption for motif validity, the common motifs problem can be directly and efficiently solved using the above algorithms.

Definition 1. *A valid motif is called* strictly valid iff *it occurs* exactly*, at least once, in (any of) the input strings.*

Consider, for instance, the DNA alphabet $\Sigma = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$. The number of possible DNA motifs of length $k := 10$ is $|\Sigma|^k = 1{,}048{,}576$. Given a dataset with a size of ≈1 MB, the possibility that there exists a motif that is valid, but not strictly valid, is rather unlikely. In other words, given such a dataset, the possibility that there exists a pattern which does not occur exactly, at least once, in the dataset *and* it also satisfies *all* the restrictions imposed by the input parameters, is rather unlikely.

The common motifs problem (detecting strictly valid motifs) can be directly solved by solving the fixed-length approximate string-matching problem for all N^2 pairs of the N input strings. Consider, for example, the common motifs problem under the Hamming distance model. We use an array u_r for each input string s_r, such that for all $1 \leq r \leq N$, $k \leq j \leq |s_r|$, $u_r[j]$ contains the total number of strings in which motif $s_r[j - k + 1\,..\,j]$ e-occurs; we set $u_r[j] := 0$, for all $0 \leq j < k$. Array u_r, for all $1 \leq r \leq N$, can easily be computed, by computing matrix M for pair (s_r, s_t), for all $1 \leq t \leq N$. While computing matrix M, we increment $u_r[j]$ only once *iff* $\mathsf{M}[i, j] \leq e$, for some $k \leq i \leq |s_t|$; as soon as we have computed the N different matrices M for s_r, it suffices to iterate over array u_r and report $s_r[j - k + 1\,..\,j]$, for all $k \leq j \leq |s_r|$, as a strictly valid motif *iff* $u_r[j] \geq q$. An array v_r, such that $v_r[j]$, for all $1 \leq j \leq |s_r|$, denoting the total number of e-occurrences of motif $s_r[j-k+1\,..\,j]$ in $s_1, \ldots, s_N$ can also be maintained. Maintaining arrays u_r and v_r does not induce additional costs. Therefore, the common motifs problem can be solved in time $\mathcal{O}(n^2)$ per matrix, where n is the average length of the N strings, thus $\mathcal{O}(N^2 n^2)$ in total.

Example 3. Let the input strings $s_r := \texttt{GTGAACT}$, $s_t := \texttt{GTCACGT}$, $e := 1$, $q := 2$, and $k := 3$. Further, let the current state of arrays u_r and v_r be:

$$\begin{array}{r@{\,:\,}l} j & 0\;1\;2\;3\;4\;5\;6\;7 \\ u_r[j] & 0\;0\;0\;0\;1\;0\;1\;0 \\ v_r[j] & 0\;0\;0\;0\;2\;0\;2\;0 \end{array}$$

Table 1b illustrates matrix M. Arrays u_r and v_r are:

$$\begin{array}{r@{\,:\,}l} j & 0\;1\;2\;3\;4\;5\;6\;7 \\ u_r[j] & 0\;0\;0\;1\;2\;0\;2\;1 \\ v_r[j] & 0\;0\;0\;1\;3\;0\;3\;1 \end{array}$$

and so the strictly valid motifs are $s_r[2\,..\,4] = \texttt{TGA}$ and $s_r[4\,..\,6] = \texttt{AAC}$.

4 Implementation

`MoTeX` is implemented as a programme that solves the common motifs problem for strictly valid motifs. `MoTeX` was implemented in the C programming language under a GNU/Linux system. It is distributed under the GNU General Public License (GPL). The open-source code and documentation are available at http://www.exelixis-lab.org/motex. The mandatory input parameters are:

– a file with the N input strings in FASTA format (sequences);
– the length k for the motifs;
– the distance d ($d := 0$ for Hamming distance *or* $d := 1$ for edit distance);
– the maximal allowed distance e;
– the quorum $q' \leq 100$ (%) as the ratio of quorum q to N.

Given these parameters, `MoTeX` outputs a text file containing the strictly valid motifs. For each reported motif, it also outputs the total number of sequences in which the motif e-occurs at least once and the total number of its e-occurrences.

Apart from the standard CPU version, `MoTeX` comes in two HPC flavors: the OpenMP version for shared memory systems and the MPI version for distributed memory systems. The user can choose the best-suited version depending on: the total size of the input sequences; the nature of the input dataset, for instance, a few very long sequences or many relatively short sequences; the available HPC architecture; and the number $p > 1$ of available processors.

Here we focus on the case when $p \leq N$, where N is the number of input sequences; that is, we have a large number of relatively short sequences. The user can choose any of the two HPC versions. `MoTeX` evenly distributes the computation of the N^2 distinct DP matrices for the N input sequences in a straightforward manner across the p processors. Therefore, if $p \leq N$, the common motifs problem for strictly valid motifs can be solved in parallel in time $\mathcal{O}(N^2n^2/p)$.

4.1 MIC Implementation

Our MIC implementation is a parallel program that uses the SMP paradigm by offloading sections of the code to run on the coprocessor. First, we used the compiler option `offload-attribute-target` to flag every global routine and global data object in the source file with the offload attribute `target(mic)`.

The compiler supports two programming models: a *non-shared memory* model and a *virtual-shared memory model*. In our implementation, we used the non-shared memory model which is appropriate for dealing with flat data structures such as scalars, arrays, and structures that are bit-wise copyable. Data in this model is copied back and forth between the host CPU and the coprocessor around regions of the offloaded code. The data selected for transfer is a combination of variables implicitly transferred because they are lexically referenced within offload constructs, and variables explicitly listed in clauses in the pragma. Only pointers to non-pointer types are supported—pointers to pointer variables are not supported. Arrays are supported provided the array element type is a scalar or bitwise copyable structure or class—so arrays of pointers are not supported. We therefore defined the following flat data structures:

– Sequence $\mathsf{S} = s_1 s_2 \ldots s_N$ of size nN, where s_i is an input sequence, for all $1 \le i \le N$, and n is the average sequence length;
– Array L of size N, such that $\mathsf{L}[i]$ stores the length of s_{i+1}, for all $0 \le i < N$;
– Array I of size N, such that $\mathsf{I}[i]$ stores the starting position of s_{i+1} in S, for all $0 \le i < N$;
– Array U of size nN, such that $\mathsf{U}[\mathsf{I}[i] \mathinner{.\,.} \mathsf{L}[i]-1]$ stores array u (see Sect. 3 for details) of s_{i+1}, for all $0 \le i < N$;
– Array V of size nN, such that $\mathsf{V}[\mathsf{I}[i] \mathinner{.\,.} \mathsf{L}[i]-1]$ stores array v (see Sect. 3 for details) of s_{i+1}, for all $0 \le i < N$.

Then we placed the offload pragma before the code block computing the N^2 distinct DP matrices for the N input sequences. While the instruction sets for the host CPU and the coprocessor are similar, they do not share the same system memory. This means that the variables used by the code block must exist on both the host CPU and coprocessor. To ensure that they do, the pragmas use specifiers to define the variables to copy between the host CPU and coprocessor:

– `in` specifier defines a variable as strictly an input to the coprocessor. The value is not copied back to the host CPU. S, L, and I were defined by `in`;
– `out` specifier defines a variable as strictly an output of the coprocessor. U and V were defined by `out`.

Therefore it becomes obvious that S, L, I, U, and V are copied *either* back *or* forth between the host CPU and the coprocessor at most once.

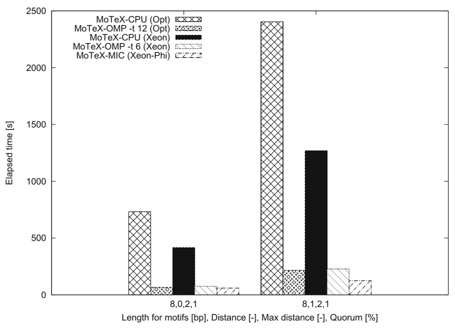
Finally, the code-block statement following the offload pragma is converted into an outlined function that runs on the coprocessor. This code is permitted to call other functions. In order to ensure that these called functions are also available on the coprocessor, we marked the functions to be called by the offloaded code block with the special function attribute `__declspec(target (mic))`.

5 Experimental Results

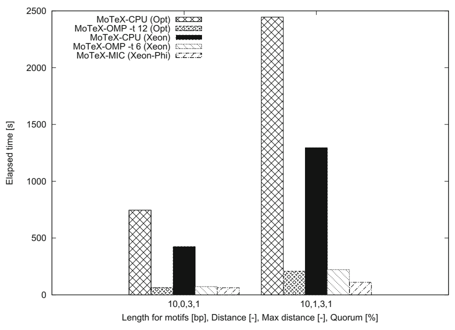
The following experiments were conducted on a GNU/Linux system running on:

– **Multicore architecture I:** a single AMD Opteron 6174 Magny-Cours CPU at 2.20 GHz with 12 cores;
– **Multicore architecture II:** a single Intel Xeon CPU E5-2630 0 at 2.30 GHz with 6 cores;
– **MIC architecture:** a single Intel Xeon host CPU E5-2630 0 at 2.30 GHz with a single Intel Xeon Phi 5110P coprocessor at 1.053 GHz with 60 cores.

We evaluated the time performance of our MIC implementation, denoted by `MoTeX-MIC (Xeon-Phi)`, running on the host CPU *and* the coprocessor against: (i) the standard CPU version of `MoTeX`, denoted by `MoTeX-CPU (Opt)`, and



(a) 1062 sequences of *Bacillus subtilis*



(b) 200 sequences of *Homo sapiens*

Fig. 1. Elapsed-time comparisons for the common motifs problem

(ii) the OpenMP version with 12 threads, denoted by `MoTeX-OMP -t 12 (Opt)`, both running on the Multicore architecture I; and (iii) the standard CPU version of `MoTeX`, denoted by `MoTeX-CPU (Xeon)`, and (iv) the OpenMP version with 6 threads, denoted by `MoTeX-OMP -t 6 (Xeon)`, both running on the Multicore architecture II. As input datasets for the programmes, we used 1062 upstream sequences of *Bacillus subtilis* genes of total size 240 KB and 200 and 1200 upstream sequences of *Homo sapiens* genes of total size 240 KB and 1.2 MB, respectively, obtained from the ENSEMBL database. We measured the elapsed time for each programme for different combinations of input parameters. In particular, we provided different values for the motif length k, the distance d used, the maximal allowed distance e, and the quorum q' as a proportion of sequences in the input dataset.

As depicted in Figs. 1 and 2, our MIC implementation accelerates the computations by a factor of ten compared to the corresponding CPU version and by a factor of two compared to the OpenMP version. We observe that, similar to other architectures such as GPGPU, for the same input dataset, the speedup gained

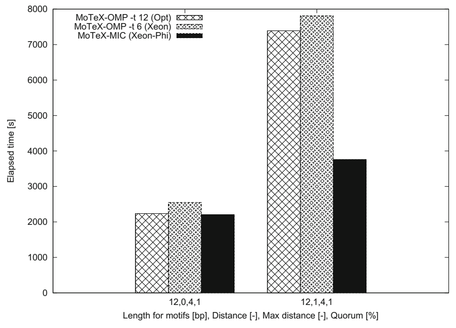


Fig. 2. Elapsed-time comparison for the common motifs problem using 1200 sequences of *Homo sapiens*

from our MIC implementation increases as the ratio of the workload to the size of the data transferred between the host CPU and the coprocessor increases. For instance, notice that in Fig. 2, while the time efficiency of `MoTeX-OMP -t 12 (Opt)`, `MoTeX-OMP -t 6 (Xeon)`, and `MoTeX-MIC (Xeon-Phi)` is similar for Hamming distance, the later programme becomes faster by a factor of two with the same dataset for edit distance, which is computationally more intensive.

References

1. Pisanti, N., Carvalho, A.M., Marsan, L., Sagot, M.-F.: RISOTTO: fast extraction of Motifs with mismatches. In: Correa, J., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 757–768. Springer, Heidelberg (2006)
2. Crochemore, M., Iliopoulos, C.S., Pissis, S.P.: A parallel algorithm for fixed-length approximate string-matching with k-mismatches. In: Elomaa, T., Mannila, H., Orponen, P. (eds.) Ukkonen Festschrift 2010. LNCS, vol. 6060, pp. 92–101. Springer, Heidelberg (2010)
3. Das, M., Dai, H.K.: A survey of DNA motif finding algorithms. BMC Bioinform. **8**(Suppl 7), S21+ (2007)
4. Iliopoulos, C.S., Mouchard, L., Pinzon, Y.J.: The Max-Shift algorithm for approximate string matching. In: Brodal, G., Frigioni, D., Marchetti-Spaccamela, A. (eds.) WAE 2001. LNCS, vol. 2141, pp. 13–25. Springer, Heidelberg (2001)
5. Lothaire, M. (ed.): Applied Combinatorics on Words. Cambridge University Press, New York (2005)
6. Marsan, L., Sagot, M.F.: Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. J. Comput. Biol. J. Comput. Mol. Cell Biol. **7**(3–4), 345–362 (2000)
7. Pavesi, G., Mereghetti, P., Mauri, G., Pesole, G.: Weeder web: discovery of transcription factor binding sites in a set of sequences from co-regulated genes. Nucleic Acids Res. **32**(Web-Server-Issue), 199–203 (2004)
8. Pissis, S.P., Stamatakis, A., Pavlidis, P.: MoTeX: a word-based HPC tool for MoTif eXtraction. In: Gao, J. (ed.) Fourth ACM International Conference on Bioinformatics and Computational Biology (ACM-BCB 2013), pp. 13–22 (2013)
9. Rombauts, S., Déhais, P., Van Montagu, M., Rouzé, P.: PlantCARE, a plant cis-acting regulatory element database. Nucleic Acids Res. **27**(1), 295–296 (1999)
10. Sagot, M.-F.: Spelling approximate repeated or common Motifs using a suffix tree. In: Lucchesi, C., Moura, A.V. (eds.) LATIN 1998. LNCS, vol. 1380, pp. 374–390. Springer, Heidelberg (1998)
11. Sinha, S., Tompa, M.: YMF: a program for discovery of novel transcription factor binding sites by statistical verrepresentation. Nucleic Acids Res. **31**(13), 3586–3588 (2003)
12. Tompa, M., Li, N., Bailey, T.L., Church, G.M., De Moor, B., Eskin, E., Favorov, A.V., Frith, M.C., Fu, Y., Kent, W.J., Makeev, V.J., Mironov, A.A., Noble, W.S., Pavesi, G., Pesole, G., Regnier, M., Simonis, N., Sinha, S., Thijs, G., van Helden, J., Vandenbogaert, M., Weng, Z., Workman, C., Ye, C., Zhu, Z.: Assessing computational tools for the discovery of transcription factor binding sites. Nat. Biotechnol. **23**(1), 137–144 (2005)

A Parallel, Distributed-Memory Framework for Comparative Motif Discovery

Dieter De Witte[1], Michiel Van Bel[2,3], Pieter Audenaert[1], Piet Demeester[1], Bart Dhoedt[1], Klaas Vandepoele[2,3], and Jan Fostier[1](✉)

[1] Department of Information Technology (INTEC), Ghent University - iMinds, Gaston Crommenlaan 8 bus 201, Ghent, Belgium
{dieter.dewitte,pieter.audenaert,piet.demeester, bart.dhoedt,jan.fostier}@intec.ugent.be
[2] Department of Plant Systems Biology, VIB, Technologiepark 927, Ghent, Belgium
[3] Department of Plant Biotechnology and Bioinformatics, Ghent University, Technologiepark 927, Ghent, Belgium
{michiel.vanbel,klaas.vandepoele}@psb.vib-ugent.be

Abstract. The increasing number of sequenced organisms has opened new possibilities for the computational discovery of cis-regulatory elements ('motifs') based on phylogenetic footprinting. Word-based, exhaustive approaches are among the best performing algorithms, however, they pose significant computational challenges as the number of candidate motifs to evaluate is very high. In this contribution, we describe a parallel, distributed-memory framework for *de novo* comparative motif discovery. Within this framework, two approaches for phylogenetic footprinting are implemented: an alignment-based and an alignment-free method. The framework is able to statistically evaluate the conservation of motifs in a search space containing over 160 million candidate motifs using a distributed-memory cluster with 200 CPU cores in a few hours. Software available from http://bioinformatics.intec.ugent.be/blsspeller/

Keywords: Motif discovery · Phylogenetic footprinting · Parallel computing · Distributed-memory

1 Introduction

Over the past decade, numerous computational methods have been proposed for the discovery of cis-regulatory elements (so-called 'motifs') in genomic sequences [1]. The most simple approaches are based on the notion of overrepresentation, i.e., the observation that a specific word occurs more often in a DNA sequence than would be expected by chance. However, as motifs are typically short and degenerate, it is difficult to discriminate between true, functional motifs and background noise.

As more and more organisms are being sequenced, methods based on phylogenetic footprinting are becoming increasingly attractive. The underlying idea is that functional regions in the DNA are subjected to selective pressure, conserving them over long evolutionary distances, in contrast to non-functional DNA which can diverge freely [2]. Such methods can therefore detect cis-regulatory elements with increased sensitivity through the identification of conserved sequences.

We developed a word-based methodology for comparative motif discovery (unpublished). Whereas most other methods restrict themselves to a (promising) subset of candidate motifs [3,4], our method is exhaustive, i.e., all words (up to a prespecified length and expressed in a degenerate alphabet) that appear in the input sequences are scored for conservation. Additionally, whereas most existing tools rely on pre-generated multiple sequence alignments [5–8] (MSA) to select conserved motif instances, our method has the ability to run in both alignment-based and alignment-free mode. The alignment-free mode has the advantage that it can detect conserved motif instances, even for distantly related species for which the generation of meaningful multiple sequence alignments is difficult.

In this contribution, we discuss the parallel framework for comparative motif discovery in which these methods were implemented. The main motivation for the development of such framework, is to make effective use of a large, distributed-memory cluster in order to deal with the increased computational requirements inherent to word-based phylogenetic footprinting. More specifically, the computational requirements are high because:

1. Our method exhaustively scores *all* words (up to a prespecified length) that are present in the input sequences. The advantage of this exhaustive approach is that optimal and complete results are produced (as opposed to statistical methods which yield a limited number of local optima, e.g. MEME [9]).
2. We allow candidate motifs to be expressed in a degenerate alphabet, leading to a combinatorial increase of the number of words to evaluate. Currently, we allow words to be expressed in a five-letter alphabet (A, C, G, T and N), however, more sensitive results could be obtained by allowing for two-fold degenerate characters or even the full IUPAC alphabet, again at the cost of increased computational requirements.
3. In order to avoid the recomputation of previously generated intermediate results, we keep all words and their conservation information in memory. Because of the large number of words, the memory requirements can exceed what can be provided by a typical workstation, advocating the development of a distributed-memory implementation.
4. In our current studies, we use four related organisms. The current trend is to incorporate more and more organisms, as this may again improve the sensitivity of the method.
5. In order to assess whether a word is significantly conserved, we employ the genome-wide statistical evaluation that was introduced in [10]. This gives rise to data dependencies between processes and hence, significant inter-process communication. By carefully choosing how the data is partitioned across the local memories of the different machines, communication volumes can be minimized.

Additional motivation can be found by looking at the evolution of computer hardware: since the introduction of the first multi-core CPU around 2003, computational power of a CPU chip has mainly progressed by incorporating more and more CPU cores. Additionally, powerful clusters are created by assembling a large number of workstations and connecting them with an intercommunication network. In order to take advantage of such hardware configurations, parallel software methodologies must be developed.

This paper is organized as follows: in Sect. 2, the framework's workflow is discussed, along with a discussion of how 'conservation' is quantified and how the genome-wide statistical testing is performed. In Sect. 3, the parallel, distributed-memory implementation is described. Section 4 presents results and current limitations of the proposed framework, followed by a conclusion and future research directions in Sect. 5.

Our parallel framework is open-source and can be obtained free of charge from http://bioinformatics.intec.ugent.be/blsspeller.

2 Comparative Motif Discovery Framework

Consider a number of S related species for which orthologous genes are grouped into so-called gene families. The promoter sequences upstream of the genes are extracted. The assumption underlying phylogenetic footprinting is that the genes within a family are regulated by the same (set of) transcription factors. In its most simple form, each of the F different gene families contains a single promoter sequence (2 strands) from each organism. The input hence consists of $2SF$ different promoter sequences[1] with a total length of $N = 2SFN_s$ characters, where N_s denotes the length of a single promoter sequence.

The framework consists of two phases: the intra-family and inter-family phase. During the intra-family step, words are scored for conservation *within* each family individually; in the inter-family step, a confidence score is established for each word w by comparing the number of gene families in which w is conserved to a background model. This statistical model was adopted from [10]. Both phases are now described in more detail.

Intra-family phase. During this step, each gene family is processed individually. Given a specific gene family, all words with a length between $l_{\min}$ and $l_{\max}$ that occur within that family are exhaustively enumerated and scored for conservation. The outcome for each word w is binary: either it is sufficiently conserved within that family or it is not. The framework imposes no restrictions on what kind of conservation metric is used. The goal of this phase is to count the number of gene families in which each word w is conserved.

In our software, we provide two complementary ways of doing this. In the *alignment-based mode*, we rely on pre-generated MSAs of the promoter sequences in a family. Words are then enumerated by adopting a sliding window approach and conservation is based on whether a word is aligned in several species within the MSA. Alternatively, in the *alignment-free* mode, a generalized suffix tree

[1] Note that we also take paralogous genes into account, hence our dataset is actually slightly larger than described.

(GST) [11] is constructed [12] from the sequences within a family, and the speller algorithm [13,14] is used to enumerate all words. The degree of conservation of a word w is based only on the presence or absence of w in a promoter sequence, regardless of the (relative) position or orientation of w. The alignment-free mode is especially beneficial for organisms that are more diverged, for which the generation of MSAs is difficult or even impossible.

In both approaches, the degree of conservation of a given word w is quantified in a biologically meaningful way, by means of the branch length score (BLS) [10]. The BLS ranges between 0 % (not conserved at all) to 100 % (fully conserved in all sequences) and takes the phylogenetic relationships between the organisms into account. If the BLS exceeds a certain threshold T, the word w is assumed to be *conserved* in that gene family.

Every conserved word w is stored in a hash table, along with the number of gene families F_w in which w is conserved. The hash table hence consists of a large number of $< w, F_w >$ key-value pairs. Words that are not conserved are not stored in the table.

Inter-family phase. During this step, for each word w that is stored in the hash table, it is established whether or not this word is significantly conserved in the complete dataset, i.e., all gene families. A confidence score C is established by comparing the number of gene families F_w in which w is conserved to the median number of gene families F_{bg} in which random permutations of w are conserved as follows:

$$C = \left[1 - \frac{F_{bg}}{F_w}\right] \tag{1}$$

Stated more precisely, given a word w, the framework generates a large number (default value = 1000) of random permutations of w and establishes the number of gene families in which these random permutations are conserved. F_{bg} then denotes the median (or representative) value. Note that all information needed to calculate the background model has already been generated during the intra-family step and can be retrieved by simple lookup operations in the hash table. The background model F_{bg} can be seen as the expected number of gene families in which a word with the same length and character composition will be conserved. If the candidate motif w is conserved in many more families that what could be expected by chance, a high confidence C will be obtained. All words w with a confidence C that exceeds a threshold (default value = 90 %) are retained and are considered true motifs.

The framework allows for the use of several conservation thresholds T_i ($i = 1 \ldots t$) in a single run. In that case, the hash table stores $< w, \overline{F_w} >$ pairs, where $\overline{F_w}$ now denotes a vector that holds the number of gene families in which w is conserved for each of the different thresholds T_i separately. A confidence value C is then obtained as the maximum confidence calculated over all thresholds T_i.

$$C = \max_{i=1\ldots t}\left[1 - \frac{\overline{F_{bg}}[i]}{\overline{F_w}[i]}\right] \tag{2}$$

The use of different thresholds provides for the detection of motifs that are significantly conserved in only a subset of the species (and thus reduced conservation threshold) in a single run and hence computationally efficient manner.

3 Distributed-Memory, Parallel Implementation

For realistic datasets, the sequential algorithm described in the previous section is computationally demanding. Even though the total number of words N_w to consider scales linearly with the total input size ($N_w = O(N)$), the number of words to consider is huge. This results in very large runtimes for the sequential algorithm.

Each word w that is conserved in at least one gene family is stored in random access memory, along with the number of gene families F_w in which it is conserved, and this for a number of BLS thresholds. The clear advantage of this approach is a strong reduction in runtime during the calculation of the background model for each motif (see Sect. 2). The disadvantage is that the memory requirements exceed what can be typically provided by a single workstation. A parallel, distributed-memory framework (see Fig. 1) was developed to alleviate both the runtime and memory bottlenecks.

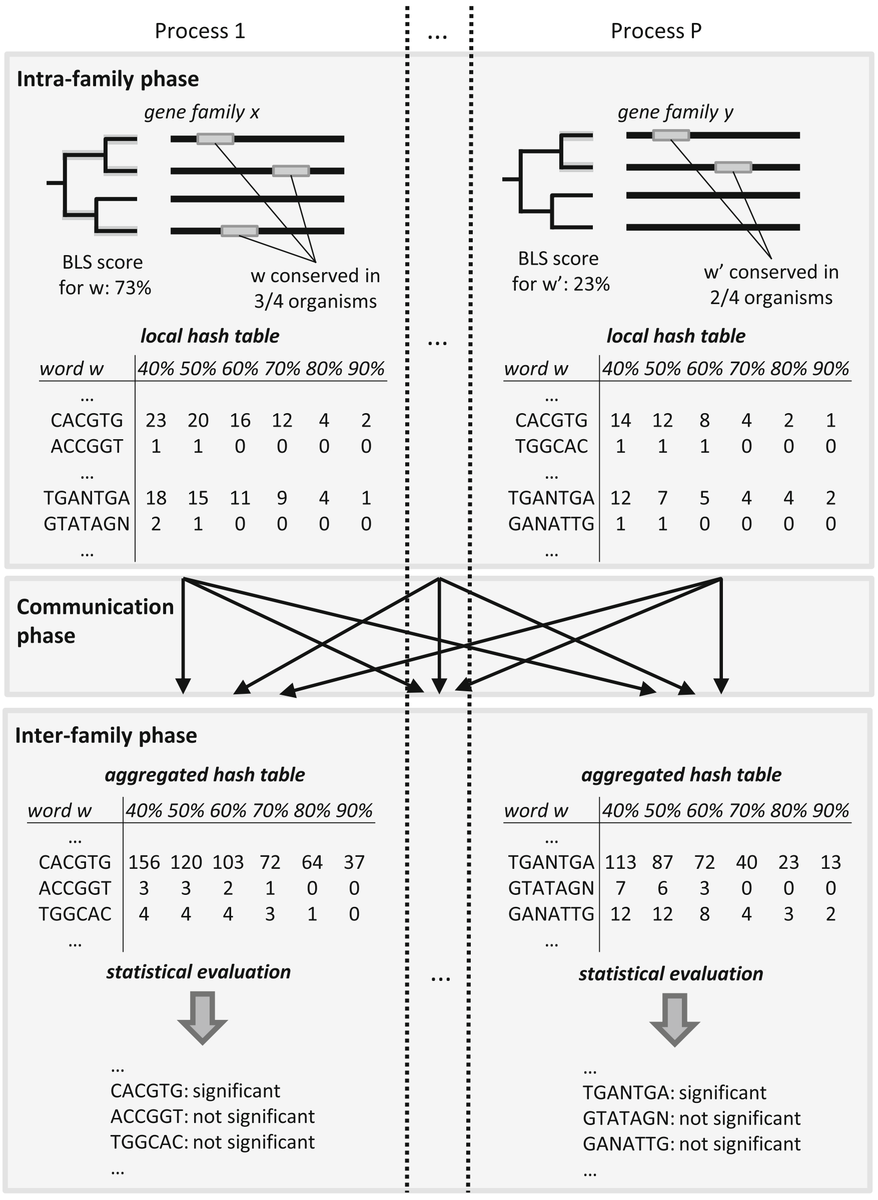
In the intra-family phase, the different gene families are uniformly distributed among the different parallel processes. Each process hence handles a subset of the gene families, and has a local hash table in which its $< w, \overline{F_w} >$ pairs are stored. This step is communication-free. At the end of this phase, a given word w can be contained by several processes, each holding only partial values in their respective $\overline{F_w}$ vectors.

In a single communication phase, these partial $\overline{F_w}$ vectors are accumulated. This is achieved by redistributing all words over the different processes, such that corresponding words are sent to the same process. That process can then sum the partial $\overline{F_w}$ vectors for each word w. Additionally, we partition the different words among the local memories of the different processes in such way that a given word and its permutations end up in the same process. For example, both the word w = CACGTG and w' = AGTGCC belong to the same *permutation group* and will end up in the same process. More specifically, for each word w, a hash value h is computed that only depends on the character composition of w, but not on the order of the characters within w. The words CACGTG and AGTGCC hence yield the same hash value h. This value is used to determine the process to which these words will be sent.

In order to obtain a uniform workload distribution during the inter-family phase, we assign a weight W_g to each permutation group g that corresponds to the maximum number of words represented by this permutation group:

$$W_g = \frac{(n_A + n_C + n_G + n_T + n_N)!}{n_A! \, n_C! \, n_G! \, n_T! \, n_N!},$$

where n_X denotes the number of characters X in a word of the permutation group. This weight is used to attribute roughly the same number of words to each process.



local hash table (Process 1)

word w	*40%*	*50%*	*60%*	*70%*	*80%*	*90%*
...						
CACGTG	23	20	16	12	4	2
ACCGGT	1	1	0	0	0	0
...						
TGANTGA	18	15	11	9	4	1
GTATAGN	2	1	0	0	0	0
...						

local hash table (Process P)

word w	*40%*	*50%*	*60%*	*70%*	*80%*	*90%*
...						
CACGTG	14	12	8	4	2	1
TGGCAC	1	1	1	0	0	0
...						
TGANTGA	12	7	5	4	4	2
GANATTG	1	1	0	0	0	0
...						

aggregated hash table (Process 1)

word w	*40%*	*50%*	*60%*	*70%*	*80%*	*90%*
...						
CACGTG	156	120	103	72	64	37
ACCGGT	3	3	2	1	0	0
TGGCAC	4	4	4	3	1	0
...						

...
CACGTG: significant
ACCGGT: not significant
TGGCAC: not significant
...

aggregated hash table (Process P)

word w	*40%*	*50%*	*60%*	*70%*	*80%*	*90%*
...						
TGANTGA	113	87	72	40	23	13
GTATAGN	7	6	3	0	0	0
GANATTG	12	12	8	4	3	2
...						

...
TGANTGA: significant
GTATAGN: not significant
GANATTG: not significant
...

Fig. 1. Schematic overview of the parallel framework. In the intra-family step, the different gene families are distributed among the P different parallel processes and each process independently scores all words within those families for conservation. A local hash table stores all $< w, F_w >$ pairs which indicate in how many gene families w has been conserved. During a single communication step, words are shuffled between parallel processes such that a given word and its permutations end up in the same process. In the inter-family step, partial F_w vectors are aggregated and for each word w, the significance of conservation is determined.

During the inter-family phase, the confidence values C are computed for each word w. This step can again be performed in parallel, as each process now holds different words. Note that because of the particular distribution of words during the previous step, this phase is now communication-free. Indeed, for every word w, all random permutations of w that are conserved are stored in the hash table of the same process. A certain random permutation that is not found in the local hash table, is not conserved in any gene family. To speed up the confidence calculations, only a single background model per permutation group is computed to which the candidate motifs are compared.

4 Results and Current Limitations

The framework was implemented in C/C++ and the Message Passing Interface (MPI) was used to handle the inter-node communication. All runs were performed on a computer cluster consisting of 25 nodes, each node containing two quad-core Intel Xeon L5420 CPUs and 16 GByte of RAM each (200 CPU cores and 400 GByte of RAM in total). The nodes communicate through a QDR Infiniband high-speed interconnection network.

As a dataset, we consider four monocot plant species: *Oryza sativa ssp. indica*, *Brachipodium distachyon*, *Sorghum bicolor* and *Zea mays*. Using the 'integrative orthology viewer' in PLAZA 2.5 [15,16], the orthology relationships between these grasses were inferred. We extracted two datasets, one with an upstream promoter length $N_s = 500$ bp, and a second dataset with a promoter length $N_s = 2$ kbp. In total, the dataset consists of $F = 17\,724$ gene families and 163 064 regulatory sequences (counting both forward and reverse strands). All words that exist in these datasets were exhaustively enumerated. Words are expressed in a 5-letter degenerated alphabet: the four bases A, C, G, T and the N character. A maximum of three 'N' characters were allowed per word. Both datasets were run using the alignment-based and alignment-free mode. Table 1 provides details for each run.

Clearly, the alignment-free mode is computationally more intensive than the alignment-based mode. This is because the definition of 'conservation' during the intra-family step is much more relaxed in the alignment-free mode, hence yielding a much higher number of words that are found to be conserved and stored in the hash table. However, because the same definition of 'conservation' is used for both the candidate word w and its control motifs (i.e. random permutations used to build the background model), the statistical test during the inter-family phase is consistent, filtering only those motifs that are conserved in a much higher number of families than expected in a random dataset. In the remainder of this Section, we focus on the computational issues. We discuss the alignment-free run on the 500 bp dataset.

During the intra-family phase, each process scores the conservation of each word in a subset of all families. Using 96 parallel processes, each process is attributed roughly 185 families. This step takes 48 minutes or 85 % of the total runtime. The load was found to be well-balanced, as the required time per family

Table 1. Motif discovery on four monocotyldon species. The dataset consists of 17 724 gene families. AF = alignment-free, AB = alignmnent-based.

Promoter length N_s	Modus	Number of parallel processes P	Runtime (walltime)	Number of words per family (avg.)	Number of sign. motifs
500 bp	AB	96	0h12	18 150	865 512
500 bp	AF	96	0h57	178 000	1 356 004
2 kbp	AB	96	0h30	28 000	902 983
2 kbp	AF	200	4h25	628 000	1 689 998

is roughly equal for each family individually. Within this step, only 6 % of the time is spent on the initial construction of the GSTs whereas 94 % of the time is spent in the discovery algorithm and the computation of the BLS values.

In the communication phase, all words are repartitioned among the different processes in such way that both a given word w and all permutations of w that were found during the intra-family step are attributed to the same process. The frequency vectors $\overline{F_w}$ corresponding to the same words are immediately merged to limit the memory overhead. The total time for this step is 8 minutes time or 14 % of the total runtime. The actual time spent redistributing the data is 5 minutes or 9 % of the total runtime, while the remaining time is spent on the packing and unpacking of the motif frequency vectors and the merging of corresponding motifs. Note that we use a high-speed Infiniband interconnection network, but that the use of an Ethernet network is also possible, as this step has only a limited contribution to the total runtime.

The inter-family phase is again communication-free and consist of the statistical testing of all words w. It required only 20 s or 0.6 % of the total runtime.

Because no computations are duplicated in the parallel algorithm, and because the single communication step has only a limited contribution to the total runtime, we expect our algorithm to exhibit a significant speedup, compared to the sequential algorithm. Note that we cannot process the complete dataset on a single node, making it difficult to estimate the exact speedup. For smaller datasets however, we achieve a speedup of up to 120, using 256 parallel processes.

The main limitation of the framework however, lies in the increased memory requirements, compared to the sequential algorithm. Whereas the sequential algorithm requires only a single $< w,\overline{F_w} >$ pair for each individual word w, the parallel algorithm has additional memory requirements because several $< w,\overline{F_w} >$ pairs might be stored in the local memories of different parallel processes. This is the case when w is found to be conserved in several gene families, contained by different processes. Therefore, the aggregated memory requirements at the end of the intra-family step are higher than in the sequential algorithm. For the largest simulation (2 kbp promoters and alignment-free mode), each of the 200 processes required almost 2 GByte of memory, yielding an aggregated memory requirement of roughly 400 GByte. Currently, this is the main limitation of the framework.

5 Conclusion and Future Research directions

In this contribution, we have presented a parallel framework for comparative motif discovery. The framework is word-based and gene-centric, as it takes a number of orthologous promoter sequences from related species as input. A measure of conservation can be defined in a flexible way. The framework allows for different alphabets (e.g. 4-letter alphabet, 4-letter alphabet + 'N' character, or even the full IUPAC alphabet) and provides for a statistical evaluation of candidate motifs based on count statistics. The framework can take advantage of large distributed-memory clusters in order to deal with high computational requirements.

Within this parallel framework, we have implemented two methodologies. First, an alignment-based approach where conservation is scored based on pre-generated multiple sequence alignments and second, an alignment-free approach where conservation does not depend on the relative position or orientation of the candidate motif. In both cases, the branch length score (BLS) was used to quantify conservation, taking the phylogenetic relationships between the organisms into account. Using this framework, we exhaustively processed four plant species.

The framework is implemented using the Message Passing Interface (MPI), but bears some conceptual resemblance with the map-reduce paradigm, where two compute phases are effectively separated by a single communication step. The framework can undoubtedly be cast in e.g. Hadoop's map-reduce implementation. The advantage of using such scheme, is that Hadoop provides for an automatic load balancing of both map and reduce phase and can recover from node failures. More importantly, map-reduce can operate out-of-core, streaming data to local hard disks if the local memory capacities turn out to be insufficient. This should, in turn allow for the handling of a larger number of organisms, or provide for more sensitive alphabets (e.g. the full IUPAC alphabet), with the ultimate goal of obtaining a comparative motif discovery method with increased sensitivity.

Acknowledgments. This work was carried out using the Stevin Supercomputer Infrastructure at Ghent University, funded by Ghent University, the Hercules Foundation and the Flemish Government - department EWI. This research fits in the Multidisciplinary Research Partnership of Ghent University: Nucleotides to Networks (N2N).

References

1. Das, M.K., Dai, H.-K.: A survey of DNA motif finding algorithms. BMC Bioinform. **8**(Suppl 7), S21 (2007)
2. Blanchette, M., Tompa, M.: Discovery of regulatory elements by a computational method for phylogenetic footprinting. Genome Res. **12**(5), 739–748 (2002)
3. Elemento, O., Tavazoie, S.: Fast and systematic genome-wide discovery of conserved regulatory elements using a non-alignment based approach. Genome Biol. **6**(2), R18 (2005)

4. Wu, J., Sieglaff, D.H., Gervin, J., Xie, X.S.: Discovering regulatory motifs in the Plasmodium genome using comparative genomics. Bioinformatics **24**(17), 1843–1849 (2008)
5. Sieglaff, D.H., Dunn, W.A., Xie, X.S., Megy, K., Marinotti, O., James, A.A.: Comparative genomics allows the discovery of cis-regulatory elements in mosquitoes. Proc. Natl. Acad. Sci. **106**(9), 3053–3058 (2009)
6. Kumar, L., Breakspear, A., Kistler, C., Ma, L.J., Xie, X.: Systematic discovery of regulatory motifs in Fusarium graminearum by comparing four Fusarium genomes. BMC Genomics **11**, 208 (2010)
7. Ettwiller, L., Paten, B., Souren, M., Loosli, F., Wittbrodt, J., Birney, E.: The discovery, positioning and verification of a set of transcription-associated motifs in vertebrates. Genome Biol. **6**(12), R104 (2005)
8. Xie, X., Lu, J., Kulbokas, E.J., Golub, T.R., Mootha, V., Lindblad-Toh, K., Lander, E.S., Kellis, M.: Systematic discovery of regulatory motifs in human promoters and 3' UTRs by comparison of several mammals. Nature **434**(7031), 338–345 (2005)
9. Bailey, T.L., Bodén, M., Buske, F.A., Frith, M., Grant, C.E., Clementi, L., Ren, J., Li, W.W., Noble, W.S.: MEME SUITE: tools for motif discovery and searching. Nucleic Acids Res. **37**, 202–208 (2009)
10. Stark, A., Lin, M.F., Kheradpour, P., Pedersen, J.S., Parts, L., Carlson, J.W., Crosby, M.A., Rasmussen, M.D., Roy, S., Deoras, A.N., et al.: Discovery of functional elements in 12 Drosophila genomes using evolutionary signatures. Nature **450**(7167), 219–232 (2007)
11. Gusfield, D.: Algorithms on Strings, Trees, And Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)
12. Giegerich, R., Kurtz, S., Stoye, J.: Efficient implementation of lazy suffix trees. Softw. Pract. Exp. **33**(11), 1035–1049 (2003)
13. Marsan, L., Sagot, M.F.: Algorithms for extracting structured motifs using a suffix tree with application to promoter and regulatory site consensus identification. J. Comput. Biol. **7**(3/4), 345–360 (2000)
14. Marschall, T., Rahmann, S.: Efficient exact motif discovery. Bioinformatics **25**(12), 356–364 (2009)
15. Van Bel, M., Proost, S., Wischnitzki, E., Movahedi, S., Scheerlinck, C., Van de Peer, Y., Vandepoele, K.: Dissecting Plant Genomes with the PLAZA comparative genomics platform. Plant Physiol. **158**(2), 590–600 (2012)
16. Proost, S., Van Bel, M., Sterk, L., Billiau, K., Van Parys, T., Van de Peer, Y., Vandepoele, K.: PLAZA: a comparative genomics resource to study gene and genome evolution in plants. Plant Cell **21**, 3718–3731 (2009)

Parallel Seed-Based Approach to Protein Structure Similarity Detection

Guillaume Chapuis[1](✉), Mathilde Le Boudic - Jamin[1], Rumen Andonov[1], Hristo Djidjev[2], and Dominique Lavenier[1]

[1] INRIA/ IRISA Rennes, GenScale, Rennes, France
{guillaume.chapuis,mathilde.le_boudic-jamin, rumen.andonov,dominique.lavenier}@irisa.fr

[2] Los Alamos National Laboratory, Los Alamos, NM, USA
djidjev@lanl.gov

Abstract. Finding similarities between protein structures is a crucial task in molecular biology. Many tools exist for finding an optimal alignment between two proteins. These tools, however, only find one alignment even when multiple similar regions exist. We propose a new parallel heuristic-based approach to structural similarity detection between proteins that discovers multiple pairs of similar regions. We prove that returned alignments have $RMSDc$ and $RMSDd$ lower than a given threshold. Computational complexity is addressed by taking advantage of both fine- and coarse-grain parallelism.

Keywords: Protein structure comparison · Parallel computing · Seed-based heuristic · Alignment graph

1 Introduction

A protein's three dimensional structure tends to be better evolutionarily preserved than its sequence. Therefore, finding structural similarities between two proteins can give insights into whether these proteins share a common function or whether they are evolutionarily related. Structural similarities between two proteins are expressed by a one-to-one mapping (also called alignment) of their three dimensional representations. The quality of these alignments is crucial to correctly estimate protein functions and protein relations. Detecting the longest alignment, when comparing protein structures, is frequently modeled as finding the maximum clique [6,8], or enumerating all maximal cliques [3,10]. Both problems are NP-hard. In these approaches, cliques are looked for in so-called product (or alignment) graphs, where each edge corresponds to matching of similar internal distances (up to a user-defined threshold τ). All edges in the target cliques satisfy this condition, but exactly this requirement leads to solving NP-hard problems.

Here, we relax this condition and accept cliques such that edges correspond to matching of similar internal distances up to 2τ. For this relaxed problem we propose a polynomial algorithm and its efficient parallel implementation comparing two protein structures that guarantees to return alignments with both $RMSDc$ and $RMSDd$ less than a given threshold value, if such alignments exist. This methodology also offers the possibility to return more than one alignment for a single pair of proteins to address cases where two proteins share more than a single similar region. Our approach takes advantage of internal distance similarities among both proteins to search for an optimal transformation to superimpose their structures. To the best of our knowledge, our tool is unique in the capacity to generate multiple alignments with "good" $RMSDc$ and $RMSDd$ values. Thanks to this property, the tool is able to detect structural repetitions within a single protein and between related proteins. We do not require vertices in the alignment graph to be ordered which make our algorithm suitable for detecting similar domains when comparing multiple domained proteins. Computational complexity is addressed by extensive use of parallel computing techniques.

1.1 Alignment Graphs

Undirected graphs $G = (V, E)$ are represented by a set V of vertices and a set E of edges between these vertices. In this paper, we focus on a subset consisting of grid-like graphs, referred to as alignment graphs.

An $m \times n$ *alignment graph* $G = (V, E)$ is a graph in which the vertex set V is depicted by an $m \times n$ array T , where each cell $T[i][k]$ contains at most one vertex (i, k) from V. An example of such an alignment graph for protein comparison is given in Fig. 1.

The matching of two proteins P_1 and P_2 can be solved by analyzing an alignment graph $G = (V, E)$, where $V = \{(v_1, v_2)|v_1 \in V_1, v_2 \in V_2\}$ and V_1 (resp. V_2) is the set of atoms of interest in protein P_1 (resp. protein P_2). A vertex (i, k) is present in V only if atoms $i \in V_1$ and $k \in V_2$ are compatible. An example of incompatibility could be different electrostatic properties of the two atoms. An edge $((i, k), (j, l))$ is in E if and only if the distance between atoms i and j in protein P_1, $d(i, j)$, is similar to the distance between atoms k and l in protein P_2, $d(k, l)$. In our case, these distances are considered similar if $|d(i, j) - d(k, l)| < \tau$, where τ is a given threshold.

1.2 Relation to Protein Structure Comparison

In an alignment graph between two proteins P_1 and P_2, a subgraph with high density of edges denotes similar regions in both proteins. Finding similarities between two proteins can therefore be performed by searching the corresponding alignment graph for subgraphs with high edge density. The highest possible edge density is found in a clique, a subset of vertices that are all connected to each other.

DAST [8], for Distance-based Alignment Search Tool, aims at finding the maximal clique in an alignment graph. DAST uses alignment graphs where rows (resp. columns) represent an ordered set of atoms V_1 (resp. V_2) from protein P_1 (resp. protein P_2). A vertex (i, j) is present in the graph if and only if residues i and j belong to similar secondary structures in both proteins. An edge is

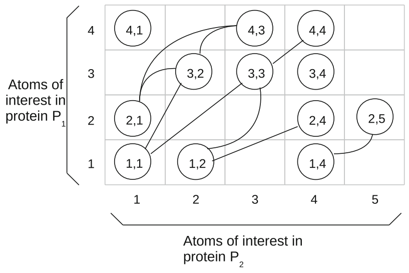

Atoms of interest in protein P$_1$
Atoms of interest in protein P$_2$
4,1
4,3
4,4
3,2
3,3
3,4
2,1
2,4
2,5
1,1
1,2
1,4

Fig. 1. Example of an alignment graph used here to compare the structures of two proteins. The presence of an edge between vertex $(1,1)$ and vertex $(3,2)$ means that the distance between atoms 1 and 2 of protein 1 is similar to the distance between atoms 1 and 3 of protein 2.

present between vertex (i,j) and vertex (k,l) if and only if $|d(i,j)-d(k,l)| < \tau$, where τ is a given threshold. By construction, alignments returned by DAST are guaranteed to have associated RMSDd strictly less than τ.

1.3 Measures for Protein Alignments

Many measures have been proposed to assess the quality of a protein alignment. These measures include additive scores based on the distance between aligned residues such as the TM-score [12] or the STRUCTAL score [11] and Root Mean Square Deviation (RMSD) based scores, such as RMSD100, SAS and GSAS [5]. Given a set of n deviations $S = s_1, s_2, ..., s_n$, its Root Mean Square Deviation is: $RMSD(S) = \sqrt{\frac{1}{n} * \sum_{i=1}^{n} s_i^2}$. Two different RMSD measures are used for protein structure comparison: $RMSDc$, which takes into account deviations consisting of the euclidean distances between matched residues after optimal superposition of the two structures; $RMSDd$, which takes into account deviations consisting of absolute differences of internal distances within the matched structures. The measured deviations are $|d(i,j) - d(k,l)|$, for all couples of matching pairs "$i \leftrightarrow k, j \leftrightarrow l$". Let P be the latter set and N_m, its cardinality. We have that $RMSDd = \sqrt{\frac{1}{N_m} * \sum_{(ij,kl)\in P} (|d(i,j) - d(k,l)|^2)}$.

2 Methods

2.1 Our Approach

Looking for the maximal clique in a graph is a NP-complete problem [4]. Being an exact solver, DAST faces prohibitively long run times for some instances. We propose a polynomial approach to protein structure comparison that guarantees to return alignments with the following properties $RMSDd < 2\tau$ and

$RMSDc < \tau$, if such exist. Our approach offers the possibility to return an arbitrary number of distinct alignments. Returning multiple similar regions can prove useful, for instance, when looking for a structural pattern that may be present more than once in a protein or when comparing highly flexible proteins. However, enumerating multiple similar regions requires a more systematic approach than those developed in other existing heuristic-based tools. The computational burden associated with such a systematic approach can nevertheless be addressed by making use of multiple levels of parallelism.

Our method is inspired by the maximal clique search implemented in DAST. Instead of testing the presence of all edges among a subset of vertices ad in DAST, we only test the presence of edges between every vertex of the subset and an initial 3-clique, referred to as seed. The correctness of the resulting algorithm follows from geometric arguments, namely that the position of any 3-dimensional solid object is determined by the positions of three of its points that are not collinear.

2.2 Overview of the Algorithm

The algorithm consists of the following three steps:

- Seeds in the alignment graph are enumerated. In our case, a seed is a set of three points in the alignment graph that correspond to two triangles (one in each protein) with similar internal distances. This step is detailed in Sect 2.3.
- Each seed is then extended. Extending a seed consists in adding all pairs of atoms, for which distances to the seed are similar in both proteins, to the set of three pairs of atoms that make up the seed. Seed extension is detailed in Sect. 2.4.
- Each seed extension is filtered. Extension filtering is detailed in Sect. 2.5 and consists in removing pairs of atoms that do not match correctly.

Filtered extensions are then ranked according to their size - number of aligned pairs of atoms - and a user-defined number of best matches are returned. This process is explained in Sect. 2.7. The overall worst-case complexity of this algorithm is $O(|V| * |E|^{3/2})$.

2.3 Seed Enumeration

A seed consists of three pairs of atoms that form similar triangles in both proteins. A triangle IJK in protein P_1 is considered similar to a triangle $I'J'K'$ in protein P_2 if the following conditions are met: $|d(I,J) - d(I',J')| < \tau$, $|d(I,K) - d(I',K')| < \tau$ and $|d(J,K) - d(J',K')| < \tau$. Here, d denotes the euclidean distance and τ is a user-defined threshold parameter. The default value for τ is 2.0 Ångstrms.

In the alignment graph terminology, these conditions for a seed ($v_i = (I, I')$, $v_j = (J, J'), v_k = (K, K')$) in graph $G(V, E)$ translate to the following: $(v_i, v_j) \in E$, $(v_i, v_k) \in E$ and $(v_j, v_k) \in E$.

A seed thus corresponds to a 3-clique in the alignment graph; i.e., three vertices that are connected to each other. Enumerating all the seeds is therefore equivalent to enumerating every 3-clique in the input alignment graph.

Not all 3-cliques, however, are relevant. Suitable 3-cliques are composed of triangles for which a unique transformation can be found to optimally superimpose them. Namely, 3-cliques composed of triangles that appear to be too "flat" will not yield a useful transformation. We thus ensure that the triangles in both proteins defined by a potential seed are not composed of aligned points (or points which are close to being aligned). The worst-case complexity of this step is $O(|E|^{3/2})$ using, e.g., the algorithms from [9].

2.4 Seed Extension

Extending a seed consists in finding the set of vertices that correspond to pairs of atoms that potentially match well (see Sect. 2.5 for details) when the two triangles defined by the seed are optimally superimposed. Finding a superset of pairs of atoms that match well is performed by triangulation with the three pairs of atoms composing the seed. The computational complexity associated to this step is $O(|V|)$.

2.5 Extension Filtering

In order to remove issues with symmetry (where the atoms in the extending pair are roughly symmetrical with respect to the plane determined by the seed atoms), we implemented a method to filter seed extensions. This method consists in computing the optimal transformation T to superimpose the triangle from the seed corresponding to the first protein onto the triangle corresponding to the second. The optimal transformation is obtained using the fast, quaternion-based method of [7]. For each pair of atoms (L,L') composing the extension of a seed, we compute the euclidean distance between $T(L)$ and L'. If the distance is greater than a given threshold τ, the pair is removed from the extension. The complexity of this step is $O(|V|)$ per seed.

2.6 Guarantees on Resulting Alignments' *RMSD* Scores

By construction, the filtering method ensures that the RMSD for a resulting alignment is less than τ: the distance between two aligned residues after superimposition of the two structures is guaranteed to be less than τ.

Internal distances between any additional pair of atoms and the seed is also guaranteed, by construction to be less than τ. Concerning internal distances between two additional pairs of atoms, we ensure that in the worst possible case, the difference is $2 * \tau$, see Fig 2. The worst possible case happens when two additional pairs of atoms $v_l = (L, L')$ and $v_m = (M, M')$, added to the extension of a seed (v_i, v_j, v_k), have atoms L, L', M and M' aligned, after superimposition, and atoms from one protein lie within the segment defined by the two other atoms. In such a case, the filtering step ensures that $d(L, L') < \tau$ and $d(M, M') < \tau$; it follows that $|d(L, M) - d(L', M')| < 2 * \tau$.

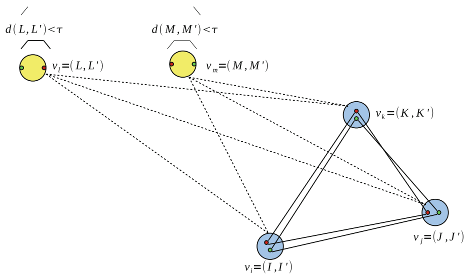


Fig. 2. Illustration of the guarantee on the similarity of internal distances between two pairs of atoms $v_l = (L, L')$ and $v_m = (M, M')$, here represented in yellow, added to a seed (v_i, v_j, v_k) represented in blue. Dashed lines represent internal distances, the similarity of which is tested in the alignment graph (color figure online).

2.7 Result Ranking

When comparing two proteins, we face a double objective: finding alignments that are both long and have low $RMSD$ scores. The methodology described in Sect. 2.5 ensures that any returned alignment will have a $RMSDd$ lower or equal to twice a user-defined parameter τ. We can therefore leave the responsibility to the user to define a threshold for $RMSD$ scores of interest. However, ranking alignments that conform to this $RMSD$ threshold simply based on their lengths is not an acceptable solution. In a given alignment graph, several seeds may lead to very similar transformations and thus very similar alignments. The purpose of returning multiple alignments for a single comparison is to find distinct similar regions in both proteins. Therefore, when two alignments are considered similar, we discard the shorter of the two.

Two alignments are considered similar, when they share defined number of pairs of atoms. This number can be adjusted depending on the expected length of the alignments or even set to a percentage of the smaller of the two compared alignments. This methodology of ranking results ensures that no two returned alignments match the same region in the first protein to the same region in the second protein.

3 Parallelism

3.1 Overview of the Implemented Parallelism

The overall complexity of our algorithm being $O(|V| * |E|^{3/2})$, handling large protein comparison with a decent level of precision - i.e., using alignment graphs with a large number of edges - can prove time-consuming. Our approach is however parallelizable at multiple levels.

Figure 3 shows an overview of our parallel implementation. Multiple seeds are treated simultaneously to form a coarse-grain level of parallelism, while a finer grain parallelism is used when extending a single seed.

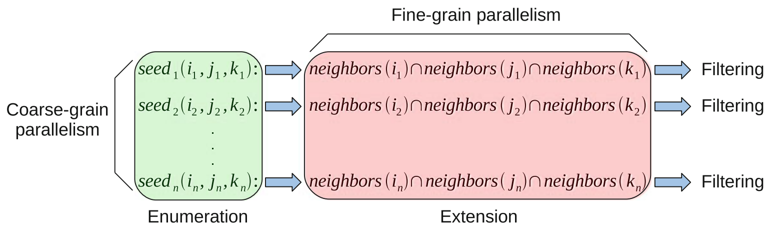

Fine-grain parallelism
Coarse-grain parallelism
$seed_1(i_1, j_1, k_1)$:
$seed_2(i_2, j_2, k_2)$:
$seed_n(i_n, j_n, k_n)$:
$neighbors(i_1) \cap neighbors(j_1) \cap neighbors(k_1)$
$neighbors(i_2) \cap neighbors(j_2) \cap neighbors(k_2)$
$neighbors(i_n) \cap neighbors(j_n) \cap neighbors(k_n)$
Filtering
Filtering
Filtering
Enumeration
Extension

Fig. 3. Overview of the implemented parallelism.

3.2 Coarse-Grained Parallelism

Computations for enumerating seeds - see Sect. 2.3, extending seeds - see Sect. 2.4, and filtering the resulting extensions - see Sect. 2.5, are independent processes, which can be performed in parallel. A user-defined number of threads can be spawned to handle, in parallel, computations for the various seeds present in the graph. This parallelism is implemented using the openMP standard [2].

Threads, however need to share their results to populate a global list of results. Inserting new entries in this global-result list would prove rather inefficient, because thread safety would need to be ensured by using locks around accesses to this result list. With such locks, threads would often stall whenever inserting a new alignment and the time lost on these accesses would only increase with the number of threads in use. In order to avoid any bottleneck when inserting a new alignment in the result list, each thread has its own private list. These lists are merged at the end of the computations to form a global result list. This method prevents the need for a synchronization mechanism and allows threads to be completely independent.

However, using this method can, in some cases, increase the total amount of computations. Whenever a seed extension is smaller than the smallest alignment present in the result list, it is discarded, thus avoiding the cost of a filtering step. Since each thread has its own result list, the minimal size required for the thread to consider filtering an extension is only a lower bound of the global minimal size found so far by all threads. Sharing only this global minimal size among threads is not a suitable solution, because no guarantee could be made on the distinctness of two alignments from different threads and smaller similar regions could be wrongly discarded.

3.3 Fine-Grained Parallelism

Seed extension makes extensive use of set intersection operations. In order to speed up these particular operations, we implemented a bit vector representation of the neighbors set of each vertex of the alignment graph. These bit vectors represent the neighbors in the alignment graph of each vertex (cf. Fig. 4). For a vertex v_i, a bit is set at position j if and only if vertices v_i and v_j are connected in the alignment graph.

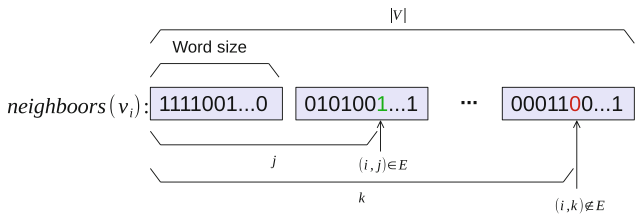


Fig. 4. Bit vector representation of the neighbors of vertex v_i in an alignment graph $G(V, E)$. In this example, v_j unlike v_k is a neighbor of v_i.

This bit vector representation of the neighbors sets allows bit parallel computations of set intersection. A simple logic *and* operation over every word element of the two sets yields the intersection.

Intersection operations also benefit from SSE[1] instructions. A number of atomic operations equal to the size of the SSE registers available on the machine (typically 128 or 256) can be computed simultaneously. However, this sparse approach to computing set intersections increases the number of atomic operations to perform. Namely, vertices, which are not neighbors of any of the two vertices for which the intersection is computed, will induce atomic operations. Such vertices would not be considered in a traditional approach to set intersection. This sparse approach is still faster in our case because alignment graphs tend to be dense enough. The size of the resulting intersections is required for the rest of our algorithm. Knowing the size of an intersection allows us to discard seeds, when larger results have already been found. Computing the size of a sparse set is not as trivial as it is with a dense set. In order to compute the size of a sparse set, we use a built-in population count instruction (POPCNT) available in SSE4. This operation returns, in constant time, the number of bits set in a single machine word. For architectures without a built-in population count instruction, a slower alternative is provided.

4 Results and Perspectives

In order to test the capacity of our approach to detect multiple regions of interest, we considered two proteins (PDB IDs 4clna and 2bbma). These proteins are each composed of two similar domains - named A and B (resp. C and D) for the first protein (resp. second protein), separated by a flexible bridge Existing approaches, such as ones based on contact map overlap (CMO) [1], tend to match both proteins integrally, yielding larger alignments but poorer RMSD scores. TM_align [13], the reference tool for protein comparison, only matches domain A onto domain C. The four top results of our tool for the comparison of these two entire proteins correspond to all four possible combinations of domain matching. Our tool was run using 12 cores of an Intel(R) Xeon(R) CPU E5645

[1] Streaming SIMD Extensions.

@ 2.40 GHz and the distance threshold was set to 7 Ångstrms and to 2 Ångstrms in the alignment graph. Scores corresponding to these alignments are displayed in Table 1.

Table 1. Details of the alignments returned by other tools - columns 2 through 4 - and our method - columns 5 through 8. Best scores are in italics.

	CMO	PAUL	TMAlign	AC	BD	AD	BC
# of aligned residues	148	148	79	72	70	66	64
% of aligned residues	100	100	53.4	48.7	47.3	44.6	43.2
RMSDc	14.781	14.781	2.935	2.048	1.731	*1.592*	2.210
RMSDd	10.838	10.838	2.627	1.797	1.475	*1.414*	1.770
TM_score	0.161	0.161	*0.422*	0.411	*0.422*	0.405	0.358

In order to test our coarse-grain parallel implementation, we compare run times obtained with various numbers of threads on a single artificially large instance. Any instance can be made artificially large by allowing a large number of vertices and edges when creating the alignment graph. The input alignment graph for this instance contains 15024 vertices for 9565358 edges. Computations were run using a varying number of cores of an Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz. Table 2 shows run times and speedups with respect to the number of CPU cores. The gain in terms of speedup becomes less significant beyond 12 cores. Note that similar results - both in terms of length and $RMSD$ scores - can be obtained in less than 30 s with a sparser alignment graph.

Table 2. Run times and speedups for varying # of cores.

# of cores	1	2	3	4	6	8	12	16	20	24
Run time (s)	6479	3696	2494	1932	1374	1072	781	723	676	643
Speedup	1	1.8	2.6	3.4	4.7	6.0	8.3	9.0	9.6	10.1

Figure 5 shows run times for graphs with a varying number of edges and the same number of vertices - 21904. Computations were run using 12 cores

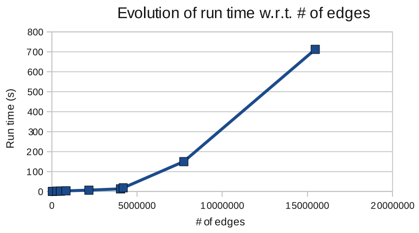


Fig. 5. Evolution of run times with respect to # of edges in the alignment graph.

of an Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz. Input alignment graphs were all generated from the same two proteins and different parameters to allow a varying number of edges.

This approach could be used to find similarities between RNA structures. However, such structures can be much larger than proteins. Therefore, future work includes further optimizations to allow larger alignment graphs to be computed.

References

1. Andonov, R., Malod-Dognin, N., Yanev, N.: Maximum contact map overlap revisited. J. Comput. Biol. **18**(1), 27–41 (2011)
2. Dagum, L., Menon, R.: OpenMP: an industry standard api for shared-memory programming. Comput. Sci. Eng. IEEE **5**(1), 46–55 (1998)
3. Gibrat, J.-F., Madej, T., Bryant, S.H.: Surprising similarities in structure comparison. Curr. Opin. Struct. Biol. **6**(3), 377–385 (1996)
4. Karp, R.M.: Reducibility Among Combinatorial Problems. Springer, Heidelberg (1972)
5. Kolodny, R., Koehl, P., Levitt, M.: Comprehensive evaluation of protein structure alignment methods: scoring by geometric measures. J. Mol. Biol. **346**(4), 1173–1188 (2005)
6. Konc, J., Janežič, D.: Probis algorithm for detection of structurally similar protein binding sites by local structural alignment. Bioinformatics **26**(9), 1160–1168 (2010)
7. Liu, P., Agrafiotis, D.K., Theobald, D.L.: Fast determination of the optimal rotational matrix for macromolecular superpositions. J. Comput. Chem. **31**(7), 1561–1563 (2010)
8. Malod-Dognin, N., Andonov, R., Yanev, N.: Maximum cliques in protein structure comparison. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 106–117. Springer, Heidelberg (2010)
9. Schank, T., Wagner, D.: Finding, counting and listing all triangles in large graphs, an experimental study. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 606–609. Springer, Heidelberg (2005)
10. Schmitt, S., Kuhn, D., Klebe, G., et al.: A new method to detect related function among proteins independent of sequence and fold homology. J. Mol. Biol. **323**(2), 387–406 (2002)
11. Subbiah, S., Laurents, D.V., Levitt, M.: Structural similarity of DNA-binding domains of bacteriophage repressors and the globin core. Curr. Biol. **3**(3), 141–148 (1993)
12. Zhang, Yang, Skolnick, Jeffrey: Scoring function for automated assessment of protein structure template quality. Proteins: Struct. Funct. Bioinf. **57**(4), 702–710 (2004)
13. Zhang, Yang, Skolnick, Jeffrey: Tm-align: a protein structure alignment algorithm based on the tm-score. Nucleic Acids Res. **33**(7), 2302–2309 (2005)

Minisymposium on Applications of Parallel Computation in Industry and Engineering

A Parallel Solver for the Time-Periodic Navier–Stokes Equations

Peter Arbenz[1](✉), Daniel Hupp[1], and Dominik Obrist[2]

[1] Computer Science Department, ETH Zürich, Zürich, Switzerland
arbenz@inf.ethz.ch
[2] Institute of Fluid Dynamics, ETH Zürich, Zürich, Switzerland

Abstract. We investigate parallel algorithms for the solution of the Navier–Stokes equations in space-time. For periodic solutions, the discretized problem can be written as a large non-linear system of equations. This system of equations is solved by a Newton iteration. The Newton correction is computed using a preconditioned GMRES solver. The parallel performance of the algorithm is illustrated.

Keywords: Time-periodic Navier-Stokes · Space-time parallelism · Nonlinear systems of equations · Newton iteration · GMRES

1 Introduction

Time-periodic problems arise in many fields. In fluid dynamics, especially with biofluids, they are of particular interest. Pulsating blood flow induced by the beating heart is probably the most prominent example. In this study we investigate, as a proof of concept, a channel flow with an immersed disk as an oscillating obstacle, see Fig. 1.

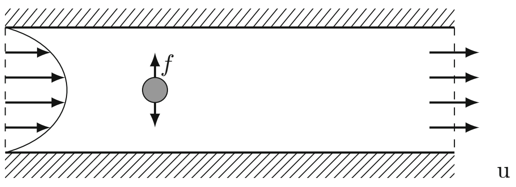


Fig. 1. Channel flow with oscillating disk.

In the classical approach to solve such problems, the transient behavior of the fluid is simulated starting from an arbitrary initial state. This simulation is continued until some periodic steady-state evolves.

Here, we model the fluid in space-time $\Omega \times [0, T)$. We will impose periodic boundary conditions in time. The discretization of the Navier–Stokes equations by finite differences in space *and* time leads to a very large nonlinear system of equations that requires parallel solution. The parallelization is done by domain decomposition where the subdomains partition space *and* time in a natural way. This is an advantage over the classical approach where only space can be partitioned. At the same time, the number of degrees of freedom is larger by $\mathcal{O}(T/\Delta t)$.

The advantages of this approach were already exploited for computing the time-periodic shallow-water equation [1]. There it could be shown that the steady-state solution can be obtained quickly, with the help of a two level circulant preconditioner. The time-periodic Burgers equation was solved in [9]. There, a truncated Fourier series is used for the time discretization instead. Also a fixed-point iteration is proposed, which allows to compute different Fourier modes in parallel.

Approaches that admit parallelization in time exist but are quite recent and not very popular yet. A task parallel approach is the *revisionist integral deferred correction method*, which is coupled with spatial domain decomposition in [2]. The parallelization in time is done by using a predictor-corrector method that allows to compute different levels of the prediction and correction in parallel. Another approach, which aims at domain decomposition in time, is *parareal* [7]. There, different time intervals are solved by different processors, given initial conditions from a cheap, serial integrator. Through iterations, the overall result gets the accuracy of the integrator done in parallel.

2 Statement of the Problem

The Navier–Stokes equation for incompressible flow can be written as

$$\partial_t \boldsymbol{u} + (\boldsymbol{u}\cdot\boldsymbol{\nabla})\,\boldsymbol{u} = -\boldsymbol{\nabla} p + \frac{1}{\mathrm{Re}}\boldsymbol{\Delta u} + \boldsymbol{f}_{\mathrm{orce}}, \qquad \boldsymbol{x}\in\Omega, \quad t>0, \tag{1a}$$

$$\boldsymbol{\nabla}\cdot\boldsymbol{u} = 0, \qquad \boldsymbol{x}\in\Omega, \quad t>0, \tag{1b}$$

$$\boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{u}_{\mathrm{bc}}(\boldsymbol{x}), \qquad \boldsymbol{x}\in\partial\Omega, \quad t>0, \tag{1c}$$

where $\boldsymbol{u}$ denotes the velocity vector, p the pressure and $\boldsymbol{f}_{\mathrm{orce}}$ an external force density. The Reynolds number Re is defined as $\mathrm{Re} = U_{\mathrm{ref}}L_{\mathrm{ref}}/\nu$, where U_{ref} and L_{ref} are reference velocity and reference length, respectively, and ν is the fluid viscosity. With $\boldsymbol{q} := [\boldsymbol{u}^T, p]^T$ we write (1) as

$$\boldsymbol{F}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{F}_{\boldsymbol{u}}^{\mathrm{force}}(\boldsymbol{q}) \\ F_p(\boldsymbol{q}) \end{bmatrix} = \begin{bmatrix} \overbrace{\partial_t \boldsymbol{u} + (\boldsymbol{u}\cdot\boldsymbol{\nabla})\,\boldsymbol{u} + \boldsymbol{\nabla} p - \tfrac{1}{\mathrm{Re}}\boldsymbol{\Delta u}}^{=\boldsymbol{F}_{\boldsymbol{u}}} - \boldsymbol{f}_{\mathrm{orce}} \\ \boldsymbol{\nabla}\cdot\boldsymbol{u} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ 0 \end{bmatrix} \tag{2}$$

The oscillating disk is modeled by an immersed boundary force formulation to enforce no-slip boundary conditions [8]. The computational domain Ω is divided into a fluid domain $\Omega_{\mathrm{f}}(t)$ and a solid obstacle domain $\Omega_{\mathrm{ob}}(t)$. The obstacle domain is changing its position in time, according to its velocity $\boldsymbol{u}_{\mathrm{ob}}(t)$. The characteristic function of the obstacle domain is defined by

$$\chi(\boldsymbol{x},t) = \begin{cases} 1, & \boldsymbol{x}\in\Omega_{\mathrm{ob}}(t), \\ 0, & \boldsymbol{x}\in\Omega_{\mathrm{f}}(t), \end{cases}$$

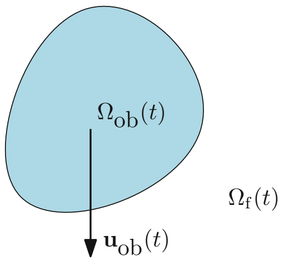


Fig. 2. Computational domain $\Omega = \Omega_{\rm f} \cup \Omega_{\rm ob}$.

which is a first order approximation of the interface $\partial\Omega_{\rm ob}(t)$ between fluid and obstacle Fig. 2. (In the actual computation we use a smoothed version of $\chi(\boldsymbol{x}, t)$, see [5].) We enforce continuity of the velocity across this interface, which we informally write as

$$\boldsymbol{u}(\boldsymbol{x}, t) = \boldsymbol{u}_{\rm ob}(\boldsymbol{x}, t) = \boldsymbol{u}_{\rm ob}(t), \qquad \boldsymbol{x} \in \partial\Omega_{\rm ob}(t).$$

We formulate the immersed boundary force

$$\boldsymbol{f}_{\rm orce} = \chi(\boldsymbol{x}, t)\boldsymbol{F}_{\boldsymbol{u}}(\boldsymbol{q}) + \chi(\boldsymbol{x}, t)(\boldsymbol{u} - \boldsymbol{u}_{\rm ob}).$$

Plugging this into the first equation of (2) yields

$$(1 - \chi(\boldsymbol{x}, t))\boldsymbol{F}_{\boldsymbol{u}}(\boldsymbol{q}) + \chi(\boldsymbol{x}, t)(\boldsymbol{u} - \boldsymbol{u}_{\rm ob}) = \mathbf{0}. \tag{3}$$

The periodicity of the oscillating obstacle induces an equal periodicity on the solution,

$$\boldsymbol{u}(\boldsymbol{x}, t) = \boldsymbol{u}(\boldsymbol{x}, t + T), \quad p(\boldsymbol{x}, t) = p(\boldsymbol{x}, t + T), \qquad \boldsymbol{x} \in \Omega,\ t > 0. \tag{4}$$

3 Discretization

The discretization is very similar to the shallow-water equation [1].

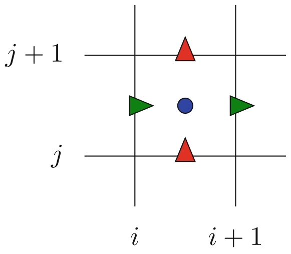


Fig. 3. Staggered grid.

We consider a space-time domain $\Omega \times [0, T]$, where $\Omega = [0, L_x] \times [0, L_y]$, which we discretize with a staggered lattice, with N_x, N_y and N_t grid points. Thus the grid points are $x_i = i\Delta x$, $y_j = j\Delta y$ and $t_k = k\Delta t$, where $\Delta x = \frac{L_x}{N_x}$, $\Delta y = \frac{N_y}{L_y}$ and $\Delta t = \frac{T}{N_t}$. The function values are approximated at grid points of a staggered grid (cf. Fig. 3)

$$\begin{aligned} u^{(k)}_{i,j+\frac{1}{2}} &\approx u(x_i, y_{j+\frac{1}{2}}, t_k),\\ v^{(k)}_{i+\frac{1}{2},j} &\approx v(x_{i+\frac{1}{2}}, y_j, t_k),\\ p^{(k)}_{i+\frac{1}{2},j+\frac{1}{2}} &\approx p(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}}, t_k). \end{aligned}$$

This leads to $N_x N_y N_t$ unknowns for p, $(N_x - 1)N_y N_t$ unknowns for u, and $N_x(N_y - 1)N_t$ unknowns for v. For discretizing (2) and (5), we use finite differences [6].

The following finite difference stencil is used for approximating the time derivatives,

$$\partial_t f^{(k)}_{i,j} \approx \frac{f^{(k-2)}_{i,j} - 4f^{(k-1)}_{i,j} + 3f^{(k)}_{i,j}}{2\Delta t}.$$

This is different to our previous work on the shallow-water equation. Furthermore, the scheme for the advective terms is changed from a central to a first order upwind scheme.

4 Numerical Solution Method

We employ Newton's iteration method to solve the nonlinear equations (2)–(3). To that end, we need the derivative of $\boldsymbol{F}$ with respect to the variable $\boldsymbol{q}$. A straightforward calculation gives [5]

$$\mathcal{D}_{\boldsymbol{q}}\boldsymbol{F}(\boldsymbol{u}) = \begin{bmatrix} \partial_t + \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}} + (\boldsymbol{u} \cdot \quad \boldsymbol{\nabla}) - \frac{1}{\mathrm{Re}}\boldsymbol{\Delta} & \boldsymbol{\nabla} \\ \boldsymbol{\nabla}\cdot & \boldsymbol{0} \end{bmatrix}, \qquad \boldsymbol{x} \in \Omega_{\mathrm{f}}, \tag{5}$$

and

$$\mathcal{D}_{\boldsymbol{q}}\boldsymbol{F}(\boldsymbol{u}) = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{\nabla}\cdot & \boldsymbol{0} \end{bmatrix}, \qquad \boldsymbol{x} \in \Omega_{\mathrm{ob}}. \tag{6}$$

Newton's method can now formally be written as in Algorithm 4.1.

Algorithm 4.1. Newton iteration for solving $\boldsymbol{F}(\boldsymbol{q}) = \boldsymbol{0}$

```
 1: Choose an initial approximation $\boldsymbol{q}^{(0)}$.
 2: for $\ell = 0, \ldots,$ maxIt-1 do
 3:    Compute $\boldsymbol{F}^{(\ell)} \leftarrow \boldsymbol{F}(\boldsymbol{q}^{(\ell)})$.
 4:    if $\|\boldsymbol{F}^{(\ell)}\|_2 < \gamma$ then
 5:       exit.
 6:    else
 7:       Solve $\mathcal{D}_{\boldsymbol{q}}\boldsymbol{F}(\boldsymbol{q}^{(\ell)})\boldsymbol{d} = -\boldsymbol{F}^{(\ell)}$ for the Newton correction $\boldsymbol{d}$.
 8:       Update $\boldsymbol{q}^{(\ell+1)} \leftarrow \boldsymbol{q}^{(\ell)} + \boldsymbol{d}$.
 9:    end if
10: end for
```

This discretization of the Navier–Stokes equations introduce above leads to the non-zero pattern of $\mathcal{D}_{\boldsymbol{q}}\boldsymbol{F}(\boldsymbol{q})$ depicted in Fig. 4. Clearly, $\mathcal{D}_{\boldsymbol{q}}\boldsymbol{F}$ has a 3×3 block structure that is inherited by the three components u, v, and p. From (3) we see that the nonzero diagonal blocks have 7 diagonals. The off-diagonal blocks have two nonzero diagonals.

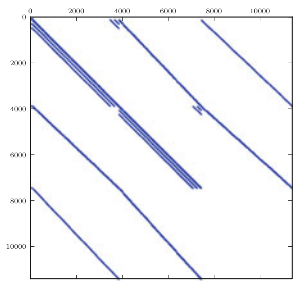


Fig. 4. Non-zero pattern for $\mathcal{D}_{\boldsymbol{q}}\boldsymbol{F}(\boldsymbol{q})$ for $N_x = N_t = 20$ and $N_y = 10$.

5 Parallelization

Our code is parallelized making use of Trilinos [4][1], a collection of numerical software packages. This object-oriented library provides numerous parallelized numerical packages. We use NOX, the package for nonlinear object-oriented solutions. It provides different line search and stepping methods for Newton methods. We will use it for its implemented backtracking and forcing methods.

We also extensively use the Epetra package that provides distributed vectors and distributed compressed row storage (CRS) matrices. They are distributed such that each processor (core) holds the variables and associated matrix rows of a block of the space-time domain. Because our space-time domain is a 3-dimensional cuboid, the best and simplest partitioning scheme is the partition in subcuboids. This reduces communication volume the most.

The matrices and vectors are distributed such that each process gets approximately the same number of rows. For $\mathcal{D}_{\boldsymbol{q}}\boldsymbol{F}$ the spatial domain is divided into m_x intervals along the x-axis and m_y intervals along the y-axis. The time domain is divided into m_t intervals. Each process gets then equations corresponding to all variables in one of these $m_x m_y m_t$ blocks. Because of this partitioning we set the number of processes equal to $m_x m_y m_t$.

The crucial step of the Newton iteration is the computation of the Newton correction $\boldsymbol{d}$ from

$$\mathcal{D}_{\boldsymbol{q}}\boldsymbol{F}(\boldsymbol{q}^{(\ell)})\boldsymbol{d} = -\boldsymbol{F}^{(\ell)}. \tag{7}$$

For applying Newton-Krylov method [10], we use the GMRES iterative solver from the Trilinos package AztecOO together with the block ILUT preconditioner [11], provided by AztecOO as well. This preconditioner constructs an incomplete LU decomposition dropping matrix entries smaller in modulus than a prescribed tolerance and allowing only a certain amount of fill-in. In this way the number of the nonzeros of the LU factors can be controlled not to exceed a certain amount relative to the original matrix. ILUT is a popular choice of preconditioner for GMRES. We set the drop tolerance $\tau = 10^{-4}$ and set the fill factor to 3, meaning that the factors have no more than three times as many nonzeros as the original matrix. Up to now, this preconditioner has showed to perform best with regard to computation time. Different perconditioners were tested, for example a circulant preconditioner similar to the one used in [1]. Also a Schur-complement solver was investigated. Because the choice of the preconditioner is crucial, more detailed investigation are planned for the future.

All computations and timing measurements were done on the Brutus cluster[2] at ETH Zurich. We used the AMD Opteron 8380 Quad-Core CPUs that are connected by an Infiniband QDR network.

[1] http://trilinos.sandia.gov/
[2] http://brutuswiki.ethz.ch/brutus/Brutus_cluster

6 Experiments

In this section, we discuss a channel flow in which the time-periodicity is introduced by a disk that oscillates with a certain frequency f (see Fig. 1).

We use a domain $\Omega = [0, L_x] \times [0, L_y] = [0, 12] \times [0, 2]$, with a Poiseuille inflow

$$u(y, t) = \frac{4}{L_y^2} (L_y - y)\, y, \qquad v(y, t) = 0.$$

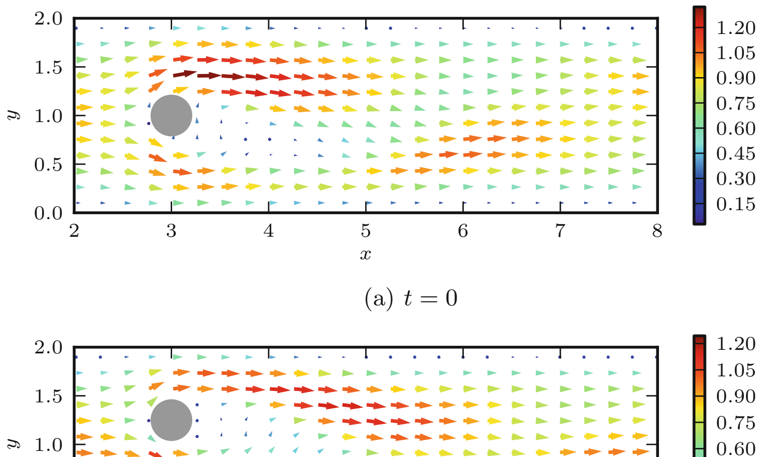
At the outflow, we drive the solution toward the same profile by a penalty function.

The reference length is half the channel width, i.e., $L_{\text{ref}} = 1$. The reference velocity is the maximum velocity at the inflow, $U_{\text{ref}} = 1$. The frequency f is chosen such that a prescribed Strouhal number $\text{St} = f\, L_{\text{ref}}/U_{\text{ref}}$ is obtained.

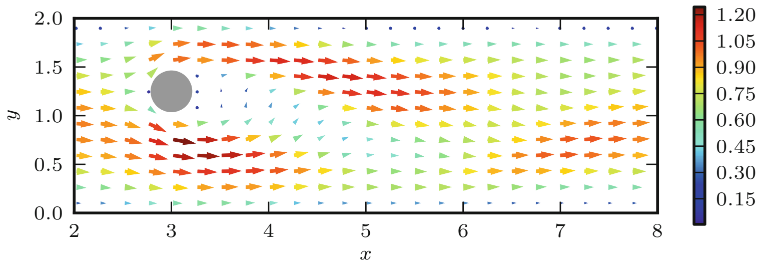
The oscillating disk has radius $L_y/10$. The center of the disk depends on time,

$$x_{\text{disk}}(t) = \frac{L_x}{4}, \qquad y_{\text{disk}}(t) = \frac{L_y}{2} + \frac{L_y}{8} \sin(2\pi f t). \tag{8}$$

We visualize a result that we obtained for $\text{Re} = 200$ and $\text{St} = 0.2$. The computational grid has $N_x = 193$, $N_y = 49$, and $N_t = 250$ grid points in the coordinate directions x, y, and t. 128 cores are employed. In Figs. 5 and 6 two snapshots of the periodic steady-state solution are shown taken at $t = 0$, and $T/4$, respectively. Figure 5 shows the velocity field, Fig. 6 the pressure field together with the



(a) $t = 0$



(b) $t = T/4$

Fig. 5. Velocity field.

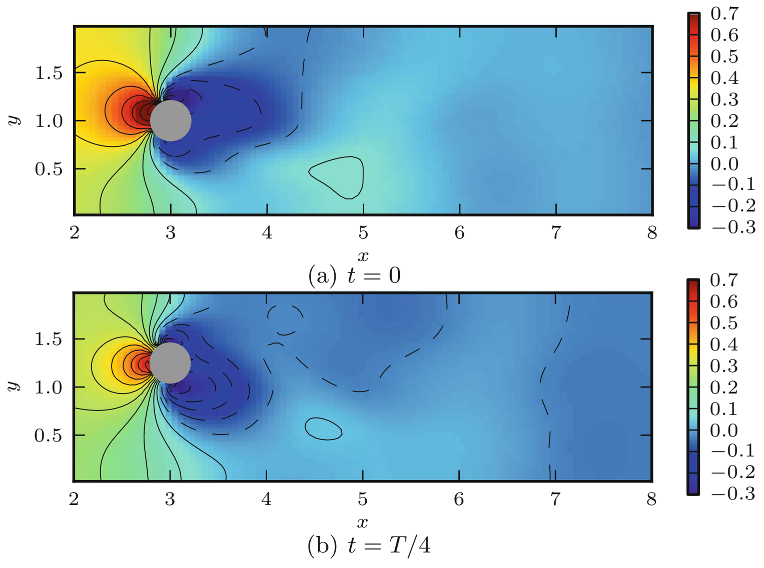


Fig. 6. Pressure field and isobars.

corresponding isobars. The boundary of the oscillating disk is well recovered by the flow field, cf. Fig. 5. Furthermore the oscillating disk leads to a laminar oscillating wake. The disturbance introduced by the oscillating disk is transported with the base flow. The pressure field in Fig. 6 shows that the highest value is obtained on the boundary of the disk, indicating a stagnation point.

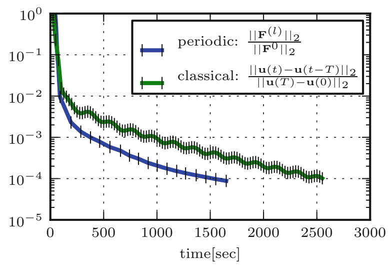


Fig. 7. Comparison between periodic and classic approach. A marker corresponds in the classical approach to the result after one computed period of physical time, in the periodic approach a dot corresponds to one Newton step.

Comparison with Time-Stepping. We compare our periodic solution approach with the classical approach, as implemented in IMPACT. IMPACT [3] is a massively parallel incompressible Navier-Stokes solver, which uses finite differences in space and a semi-implicit time integration scheme. For both approaches the same problem and the same discretization is used as before.

The performance of both approaches is compared by looking at the reduction of the initial residual over computation time. In case of the periodic approach the residual $||\boldsymbol{F}||_2$ is used. In the classical approach $||\boldsymbol{F}||_2$ is always zero, because the equations are solved explicitly. In the classical approach we use as a residual $||\boldsymbol{u}(t) - \boldsymbol{u}(t - T)||_2$, the difference between one period of time.

By construction, this difference is always zero in the periodic approach. Both are scaled with their initial residual. The reason for that is, that both start with the same initial guess: an undisturbed Poiseuille flow.

Figure 7 shows that the computation time to reduce the residual of the initial guess by four orders of magnitude needs a similar amount of computation time.

Weak Scalability. First we want to investigate how the resolution influences the performance of the algorithm. We set Re = 50 and St = 0.2 and use a grid with $N_x \times N_y \times N_t = 50i \times 25i \times 25i$, where i varies from one to four. If we choose the number of cores as $2\,i^3$ the workload per core remains constant. In Fig. 8 we can see how the performance deteriorates as the resolution increases. On the right panel we see that the ILUT preconditioner loses quality as the number of processors increases. Also, if the accuracy of the inner solves is insufficient then the number of steps in the Newton iteration increases (Fig. 8).

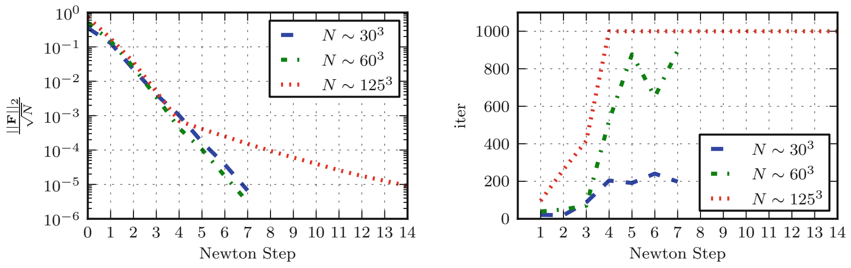


Fig. 8. Weak scaling. Left: residual norm vs. Newton step. Right: number of GMRES steps to solve for Newton correction vs. Newton step (upper limit 1000).

Speedup. For measuring the speedup we consider the same geometry as before but we set Re = 50 and St = 0.1. The computational grid is $144 \times 72 \times 72$. The problem has about 2.2 million degrees of freedom. The speedup for p processors is obtained by dividing the computation time (wall-clock time) for one processor by the computation time for p processors t_1/t_p. The efficiency is the speedup divided by number of cores.

Table 1. Performance numbers.

p	Time (min)	Speedup	Efficiency	p	Time (min)	Speedup	Efficiency
1	231	1	1.00	16	30.3	7.6	0.48
2	123	1.9	0.94	64	8.44	27.4	0.43
4	71.9	3.2	0.80	128	3.39	68.1	0.53
8	48.2	4.8	0.60				

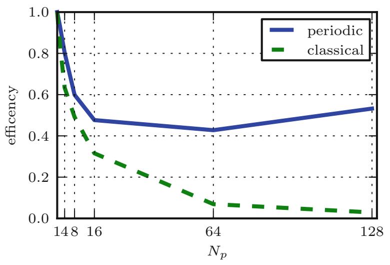


Fig. 9. Strong scaling. Efficiencies of periodic and classical approach.

In Table 1 we show execution times, speedups and efficiencies for various numbers of cores. The execution times are minima from several runs. The corresponding convergence behaviors are found in Fig. 10. From this figure it appears that the core number does not affect much the convergence behavior. However, there are differences at Newton step 4, where we have faster convergence for 2, 64, and 128 cores. Further, for $p = 2$ and $p = 128$ the Newton iteration converges after 11 steps instead of 12. This partly explains the increase of the efficiency with 128 cores.

Additionally, we compare the efficiency between the classical approach and the periodic approach. The classical approach uses the same spatial discretization as the periodic approach, but uses a smaller time step, to satisfy the CFL condition. In Fig. 9 we can see that the periodic approach scales much better then the classical approach.

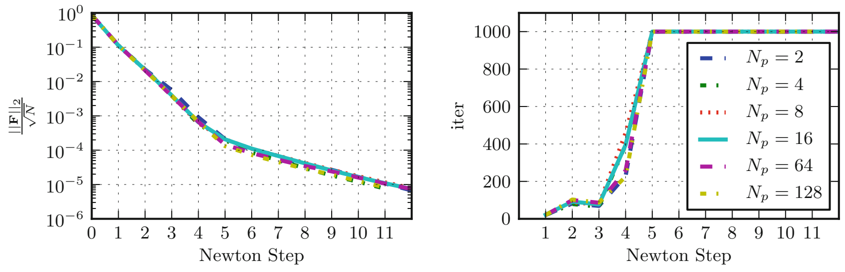


Fig. 10. Strong scaling. Left: residual norm vs. Newton step. Right: number of GMRES steps to solve for Newton correction vs. Newton step (upper limit 1000).

7 Conclusions

We have extended our earlier results on Burgers and the time-periodic shallow-water equation [1,9] to the time-periodic Navier–Stokes equations. We have replaced our time consuming, cumbersome multilevel preconditioner for the correction equation inside the Newton iteration. We now use a block-Jacobi preconditioner where the number of blocks equals the number of processors (cores). In fact, we apply only an ILUT factorization instead of a full-fledged LU factorization to save memory. The results show that the quality of the block-Jacobi preconditioner worsens as the core number increases from 1 to 128. As a result the number of inner iterations increases, and the upper limit is reached after some time. At this point also an effect on the (outer) Newton iteration is observed. With an efficiency of more than 50 % we have reduced the execution time of the discussed problem from almost 4 h on 1 core to less than 4 minutes on 128 cores.

In addition, we could show that our periodic approach scales better than the classical approach. And, although in our approach the performance of the linear solver is not optimal, the computation time to solve a problem is still not worse than solving it with the classical approach.

References

1. Arbenz, P., Hiltebrand, A., Obrist, D.: A parallel space-time finite difference solver for periodic solutions of the shallow-water equation. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011, Part II. LNCS, vol. 7204, pp. 302–312. Springer, Heidelberg (2012)
2. Christlieb, A.J., Haynes, R.D., Ong, B.W.: A parallel space-time algorithm. SIAM J. Sci. Comput. **34**(5), C233–C248 (2012)
3. Henniger, R., Obrist, D., Kleiser, L.: High-order accurate solution of the incompressible Navier-Stokes equations on massively parallel computers. J. Comput. Phys. **229**(10), 3543–3572 (2010)
4. Heroux, M.A., Bartlett, R.A., Howle, V.E., Hoekstra, R.J., Hu, J.J., Kolda, T.G., Lehoucq, R.B., Long, K.R., Pawlowski, R.P., Phipps, E.T., Salinger, A.G., Thornquist, H.K., Tuminaro, R.S., Willenbring, J.M., Williams, A., Stanley, K.S.: An overview of the Trilinos project. ACM Trans. Math. Softw. **31**(3), 397–423 (2005)
5. Hupp, D.: A parallel space-time solver for Navier-Stokes. Master thesis, ETH Zurich, Curriculum Computational Science and Engineering. http://dx.doi.org/10.3929/ethz-a-009979902, May 2013
6. LeVeque, R.J.: Finite Difference Methods for Ordinary and Partial Differential Equations. SIAM, Philadelphia (2007)
7. Lions, J.-L., Maday, Y., Turinici, G.: A "parareal" in time discretization of PDE's. C. R. Math. Acad. Sci. Paris **332**(7), 661–668 (2001)
8. Mohd-Yusof, J.: Combined immersed-boundary/B-spline methods for simulations of flow in complex geometries. Annual Research Briefs, pp. 317–327. NASA Ames/Stanford Univ., Center for Turbulence Research. http://ctr.stanford.edu/ResBriefs97/myusof.pdf (1997)
9. Obrist, D., Henniger, R., Arbenz, P.: Parallelization of the time integration for time-periodic flow problems. PAMM **10**(1), 567–568 (2010)
10. Pawlowski, R.P., Shadid, J.N., Simonis, J.P., Walker, H.F.: Globalization techniques for Newton-Krylov methods and applications to the fully coupled solution of the Navier-Stokes equations. SIAM Rev. **48**(4), 700–721 (2006)
11. Saad, Y.: ILUT: a dual threshold incomplete LU factorization. Numer. Linear Algebra Appl. **1**(4), 387–402 (1994)

Parallel Numerical Algorithms for Simulation of Rectangular Waveguides by Using GPU

Raimondas Čiegis[1](✉), Andrej Bugajev[1], Žilvinas Kancleris[2], and Gediminas Šlekas[2]

[1] Vilnius Gediminas Technical University, Saulėtekis av. 11, 10223 Vilnius, Lithuania
{rc,andrej.bugajev}@vgtu.lt
[2] Semiconductor Physics Institute, A. Goštauto 11, 01108 Vilnius, Lithuania
{kancleris,slekas}@pfi.lt

Abstract. In this article we consider parallel numerical algorithms to solve the 3D mathematical model, that describes a wave propagation in rectangular waveguide. The main goal is to formulate and analyze a minimal algorithmic template to solve this problem by using the CUDA platform. This template is based on explicit finite difference schemes obtained after approximation of systems of differential equations on the staggered grid. The parallelization of the discrete algorithm is based on the domain decomposition method. The theoretical complexity model is derived and the scalability of the parallel algorithm is investigated. Results of numerical simulations are presented.

Keywords: Parallel algorithms · Numerical simulation · Wave propagation · GPU · CUDA · Scalability analysis

1 Introduction

Some relatively new technologies have created new challenges for mathematicians and programmers in the field of parallel computing. GPU (Graphics Processing Unit), which is inside of the most video cards, is capable of doing huge numbers of FLOPS (floating point operations per second). In certain circumstances new GPUs overcome any new CPU (Central Processing Unit) in terms of performance/price. However, GPUs have many restrictions and require specific algorithms to fit these restrictions and to use GPUs resources efficiently. The most important task of efficient calculations on GPU is the implementation of the optimal data transfers among different layers of memory. That is why the main GPU computation efficiency indicator is GPU bandwidth. The bandwidth obtained in application compared to the theoretical peak bandwidth shows how effectively GPU is used.

Solving the Maxwell equations on CUDA with FDTD method was studied in many papers. For example Adams [2] and Liuge [3] both showed good speed-ups comparing to traditional CPU computations. Micikevicius [4] has implemented 3D FDTD algorithm, this work is included in CUDA Software Development Kit as an example project. However, there is a lack of mathematical formalism and techniques that would make analysis of GPU technology closer to a classical mathematical theory of parallel algorithms. Our goal is to systematize the GPU hardware engineering facts and specific properties of the parallel algorithms developed for data parallel computations into one simple but robust template of such algorithms. The optimal usage of this template for different algorithms is based on a simple theoretical performance model. This model helps the user to predict the complexity of computations and enables him to make the scalability analysis of developed parallel algorithms. Then the performance of proposed parallel algorithm can be optimized with respect to given parameters of the model [5].

2 CUDA

All the technical details that are collected in this section were taken from the official NVIDIA documents "NVIDIA CUDA C Programming Guide" and "CUDA C Best Practices Guide".

CUDA architecture. CUDA as programming model includes only the most important (in terms of computation) aspects of GPUs architecture, hiding from programmer unneeded device hardware details. Generally the whole architecture can be separated to *host processor* (i.e. CPU processor) and the device, i.e. GPU itself. The device consists of array of streaming multiprocessors (SMs). They execute blocks of threads. In general one SMs can execute more than one block of threads simultaneously.

The GPU memory is divided into the following hierarchy: (a) host memory, which is not actually a part of the device, but the device must transfer data from/to host; (b) global memory, which is accessible by all threads that run on all SMs; (c) shared memory, which is located on each SM and is accessible only to the threads of the same block. Shared memory is much faster than global memory, but it has a very limited size. There are also three more types of memory: constant, texture and local memory spaces. Constant and texture memories are used for read-only constants and they are placed on GPU cache. The local memory actually is not a separate memory space. It is an abstraction to the local scope of a thread and actually this memory is located on the global memory.

Let us overview the most important steps for any algorithm implemented on CUDA.

1. *Task parallelization.* The problem is split into blocks of SIMD (Single Instruction Multiple Data) tasks, which are performed by different threads in parallel. Array of blocks is constructed, and each block is indexed by using one-, two- or three-dimensional indexing. Tasks in a block are assigned to the threads, which are indexed inside the given block. All GPU operations

are performed by warps (32 threads) or half-warps (16 threads). Data transfers are performed by warp transactions and must satisfy data alignment requirements to be performed effectively, these requirements vary on GPUs with different compute capabilities.

2. Let us note the main aspects which are important in order to use threads efficiently. First, each SM executes blocks of tasks that are given from array of blocks. Second, a SM can simultaneously execute only a limited number of thread blocks due to the following restrictions: there is the maximum number of active warps N_{wmax}, the maximum number of registers N_{rmax}, the maximum size of shared memory and the maximum number of active blocks N_{bmax} per SM.

GPU occupancy. The GPU occupancy is defined as the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. We note that a possibility to execute some other warps when one warp is paused or stalled is the only way to hide memory latencies and keep the hardware busy. Higher occupancy does not always leads to higher performance, there is a point above which additional occupancy does not improve performance. However, low occupancy always interferes with the ability to hide memory latency, resulting in performance degradation.

It is obvious that the number of active warps depends on the execution configuration of the kernel, the memory resources of the multiprocessor, and the resource requirements of the kernel. In order to choose optimal thread block sizes, the CUDA Software Development Kit provides a simple CUDA Occupancy Calculator, which helps users to take into account register and shared memory requirements. In this paper we describe these requirements in the form, adapted to a class of explicit data parallel algorithms, when we assume that each active warp is using all 32 threads. Then the occupancy ϱ is calculated as

$$\varrho = N_t N_b / N_{tmax}, \tag{1}$$

where N_t is the number of threads in a block, N_b is the number of active blocks per SM, N_{tmax} is the maximal number of threads per SM. It is easy to see that $\varrho \in (0, 1]$. Let N_r be the number of registers used per thread, N_s is the size of shared memory per block, N_{smax}, N_{rmax} and N_{bmax} are the total shared memory, the number of registers and the maximum number of blocks per SM. Then the main restrictions for GPU resources can be written as

$$\begin{cases} N_t \le N_{tmax}, \quad N_b \le N_{bmax}, \quad N_t N_b \le N_{tmax}, \\ N_b N_s \le N_{smax}, \quad N_b N_t N_r \le N_{rmax}. \end{cases} \tag{2}$$

The value of occupancy for any implementation of data parallel algorithms can be estimated by parameters N_t, N_s and N_r. From (2) we obtain that the number of active blocks per SM should satisfy conditions

$$N_b \le \min\left(N_{bmax}, \left[N_{tmax}/N_t\right], \left[N_{smax}/N_s\right], \left[N_{rmax}(N_t N_r)\right]\right), \tag{3}$$

where $[\cdot]$ is the integer part of a real number. After we have parameters N_b and N_t, ϱ is calculated from (1). Let us discuss how to obtain the maximum occupancy value $\varrho = 1$, when all threads of SM are active. As it follows from analysis given above, it is needed to have at least two active blocks $2 \leq N_b \leq N_{bmax}$. Then from (1) and estimates of restriction on parameters we get the estimates on N_t and N_b:

$$N_t \leq N_{tmax}/2, \quad N_b = N_{tmax}/N_t. \tag{4}$$

Conditions (2), (4) give the required estimates

$$\frac{N_{tmax}}{N_{bmax}} \leq N_t \leq \frac{N_{tmax}}{2}, \quad N_s \leq N_t \frac{N_{smax}}{N_{tmax}}, \quad N_r \leq \frac{N_{rmax}}{N_{tmax}}. \tag{5}$$

Assuming that $N_{tmax}/N_t = [N_{tmax}/N_t]$, (5) are necessary and sufficient conditions for all threads to be active.

3 The Template of Numerical Algorithms

Three main goals are formulated for any parallel algorithm: to expose enough parallelism, to balance work across the SMs and to minimize data transfers with low bandwidth. Data parallel explicit numerical algorithms typically are memory bound codes, since the arithmetic intensity of such algorithms is quite low and not sufficient to hide latencies of communications. Thus they need more occupancy to get the maximum performance.

At each time step, 3D FDTD problem can be represented by sequential explicit calculations of solutions of very large sparse systems of linear equations. One iteration can be described in the compact form $M = F(A_1, A_2, \ldots, A_m)$, where $M, A_1, A_2, \ldots, A_m$ are $N_x \times N_y \times N_z$ dimension arrays. Operator F is defined on a given stencil of the grid

$$M_{ijk} = f(S_{ijk}(A_1), S_{ijk}(A_2), \ldots, S_{ijk}(A_m)),$$

where $S_{ijk}(A)$ is a subset of elements around the element $A_{i,j,k}$, described by a stencil of the approximation algorithm, f is a problem dependent function. For example, let us consider a classical cross stencil with radius $s = 1$, then S_{ijk} is defined as:

$$S_{i,j,k}(A) = \{A_{ijk}, A_{i-1,j,k}, A_{i+1,j,k}, A_{i,j-1,k}, A_{i,j+1,k}, A_{i,j,k-1}, A_{i,j,k+1}\}. \tag{6}$$

To parallelize calculations it is necessary to split the corresponding data into blocks and calculate them separately in parallel. Due to stencil requirements each block must be expanded by additional elements called *halo* elements [4].

Here we formulate the main tasks of our project. First, we are interested to investigate if the usage of shared memory can help to optimize the efficiency of the code for various types of stencils (cross stencils with different radius lengths, non-symmetrical stencils, full box stencils). Second, for devices of compute capability 2.x instead of shared memory the device can use L1 memory, that is handled automatically. It is important to compare the efficiencies of both strategies. Third, we want to test parallel algorithms implemented by OpenACC. The main benefit of the last strategy is that optimizing code with directives is quite easy compared to writing CUDA kernels. OpenACC is avoiding restructuring of existing code for production applications, especially if the parallel code is implemented in OpenMP.

Shared Memory Based Algorithms. Since shared memory on a device has a very limited size, the sizes of blocks defining subtasks should be small enough. To indicate the efficiency of data transfer, we define *redundancy* R as the ratio between the number of elements accessed and the number of elements processed. The redundancy $R = R_1 + R_2$ consists of read redundancy R_1 and write redundancy R_2. We assume that the number of elements read is N_r, the number of elements written is N_w, the number of elements processed is N_p, then the redundancy is calculated by the following formula: $R = N_r/N_p + N_w/N_p$. In ideal case the number of elements read and written are equal to the number of elements processed, then the read and write redundancies both are equal to 1 and the overall redundancy is equal to 2. However, because of a stencil we need to read halo elements, so R_1 is bigger than one for stencil-driven calculations.

3D data splitting. We split 3D data in all three directions into blocks with sizes $N_{tx} \times N_{ty} \times N_{tz}$. Since calculations involve the information on a stencil set, then to process the block of data we need to access halo elements from each side of sub-block. The total number of additional elements is equal to $2(N_{tx}N_{ty} + N_{ty}N_{tz} + N_{tx}N_{tz})s$, where s is the radius of the stencil. So for one block the overall redundancy is

$$R = 2 + 2s(N_{tx}N_{ty} + N_{ty}N_{tz} + N_{tx}N_{tz})/(N_{tx}N_{ty}N_{tz}). \tag{7}$$

2D data splitting. We split 3D data only in x and y directions into blocks with sizes $N_{tx} \times N_{ty} \times N_z$. Since the whole block can't be placed into a shared memory, we need to construct an algorithm how data should be processed. The whole block is split into sub-blocks with sizes $N_{tx} \times N_{ty} \times N_{tz}$ and sub-blocks are processed in z direction. Thus we can keep in shared memory a part of data from the previous sub-block. To be exact, the neighbour sub-block in z direction needs $2sN_{tx}N_{ty}$ elements of the current sub-block. So this sub-block needs to read only $N_{tx}N_{ty}N_{tz} + 2s(N_{ty}N_{tz} + N_{tx}N_{tz})$ elements. The overall redundancy is

$$R = \frac{2N_{tx}N_{ty}N_{tz} + 2s(N_{ty}N_{tz} + N_{tx}N_{tz})}{N_{tx}N_{ty}N_{tz}} = 2 + \frac{2s(N_{ty} + N_{tx})}{N_{tx}N_{ty}}. \tag{8}$$

We see from (8) that the redundancy does not depend on N_{tz}, so it is optimal to take $N_{tz} = 1$ in order to make N_{tx} and N_{ty} as big as possible and to minimize the redundancy (8). So the optimal sub-block becomes a slice in x and y plane.

This approach is already mentioned by Micikevičius [4]. However, he interpreted this approach in a different way – as two-pass approach, similar to one described for Cell processors. We have derived this algorithm from general data splitting techniques of parallel computation theory.

Let us summarize this algorithm. 3D data is split into blocks of sizes $N_{tx} \times N_{ty} \times (N_z - 2s)$. Thus we get a grid of blocks of the size $\dfrac{(N_x - 2s)}{N_{tx}} \times \dfrac{(N_y - 2s)}{N_{ty}} \times 1$. For each block data is processed iteratively by slices in z direction. Here we present the pseudo code for calculations

```
**for** $k = s \rightarrow N_z - s - 1$ **do**
    calculate $M_{i,j,k}\ for\, \forall i, j$ inside the block
**end for**
```

For the cross stencil halo elements in z direction can be stored in the registry memory, since they are needed for one thread only. At the first step of each block calculations, the required data is copied only for the first slice. The pseudo code of this algorithm is the following:

```
**for** $i = s \rightarrow N_z - s - 1$ **do**
    **if** $i == s$ **then**
        $sN_{tx}N_{ty}$ elements of slices $z = 0, \dots, s-1$ and $z = s+1, \dots, 2s-1$
        are copied from device to registry memory;
        $N_{tx}N_{ty} + 2s(N_{tx} + N_{ty})$ elements (with halo) of slice $z = s$
        are copied from device to shared memory;
    **else**
        1. $(i - s - 1)$-th slice is deleted from registry memory;
        2. The elements (without halo) of $(i - 1)$-th slice are copied from
        shared to registry memory, then this shared memory is freed;
        3. Shared halo elements from $(i - 1)$-th slice are deleted from
        shared memory and halo elements from $i$-th slice are copied to it
        from device;
        4. Elements from $i$-th slice are copied from registry to shared memory;
    **end if**
    5. $(i + s)$-th slice is copied from device to registry memory;
    6. Data from $i$-th slice is processed and written back to device;
**end for**
```

4 Theoretical Model for the Performance Evaluation

Simple but accurate complexity models of data parallel algorithms are constructed by using this standard form

$$T = \beta_1 N + M(\alpha + \beta_2 N + \gamma \bar{N}), \tag{9}$$

where T is computation time of the algorithm, β_1 determines time required to send one element from/to host to/from global memory, α is data communication start-up time, β_2 determines time required to send one element from/to global memory to/from shared memory, N is the number of communicating elements, γ determines the complexity of one basic computational operation, $\bar{N}$ is the number of such operations, M is the number of time steps.

In CUDA there are two main transfer operations: from host to global memory and from global memory to shared/registry/cache. Assuming that the number of time steps M is large enough we can neglect the initial and final data transfer from and to host and global memory. Also we will assume that start-up costs are negligible $\alpha = 0$, then we get a simplified performance model of one time step algorithm

$$T = \beta_2 N + \gamma \bar{N}. \tag{10}$$

For explicit data parallel algorithms considered in this paper the cost of data transfers between global and shared memory are the most important. Parameter β_2 depends on many factors such as GPU occupancy ϱ, redundancy R and some more specific aspects of algorithm implementation. We propose the following simple formula to estimate β_2

$$\beta_2 = (1 + \beta_h d_h)/W(\varrho), \tag{11}$$

where W is bandwidth, a monotonically increasing function, that is determined experimentally, $\varrho \in (0, 1]$ is GPU occupancy, β_h is experimentally determined constant, that shows the difference between transfer times of halo and regular elements

$$d_h = \frac{\text{number of transfers of arrays with halo elements}}{\text{total number of transfers of arrays}}.$$

We have assumed that (11) does not depend on R. The bandwidth W decreases due to additional stalls of threads at synchronization points that are necessary to perform shared memory management by the whole block of threads, i.e. it describes the cost of using stencil in data accessing pattern.

In order to determine the function $W(\varrho)$ we consider a simple benchmark with three input arrays A, B, C, and one output array D. One-point grid stencil $S_{i,j,k}(A) = A_{i,j,k}$ is used, so there are no halo elements. We take the kernel function F which is defined by the pseudo-code

```
$F(A_{i,j,k}, B_{i,j,k}, C_{i,j,k})$ :
Begin
$d = 1$
for $\ell = 1 \to K$ do
    $d = d + d * A_{i,j,k};\ d = d + d * B_{i,j,k};\ d = d + d * C_{i,j,k}$
end for
End
```

Here K is the number of iterations. In computational experiments we have used 3 different video cards, we will refer to them as GPU1, GPU2, GPU3:

Table 1. Implementation efficiency analysis

GPU	Bandwidth (GB/s)	b.t. peak	GFlops	GFlops t. peak
GPU1	10.4	12.8	52.2	67
GPU2	83	128	579	874
GPU3	59.7	86.4	916	1425

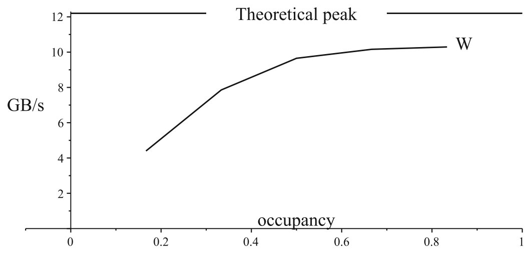


Fig. 1. The bandwidth $W(\varrho)$ for GPU1 device

- GPU1: GeForce 9400 GT, compute capability 1.1, 12.8 GB/s, 67.2 GFlops
- GPU2: GeForce GTX 460, compute capability 2.1, 128 GB/s, 1042 GFlops
- GPU3: GeForce GTX 650 Ti, compute capability 3.0, 86.4 GB/s, 1425 GFlops

In Table 1, we present the maximum values of GFlops and bandwidth (Gbytes/s) that were achieved for this benchmark. Note that more GFlops can be achieved for different benchmarks, but it is not important, since β_2 has the most important impact on calculations. For the given test problem performance model (10) can be rewritten as

$$T = \frac{4N}{W(\varrho)} + 6K\gamma N, \tag{12}$$

here $N = N_x N_y N_z$. Next we determine constant γ by fixing ϱ and changing K. All arithmetical calculations were done using 32 bit float data type.

Afterwards we fix K and change ϱ to determine $W(\varrho)$. In all the following computational experiments we use arrays sized $256 \times 256 \times 256$, the number of iterations is fixed to $K = 10$. It is worth to note, that the increase of these values has a small effect on the obtained results. The value of ϱ was controlled by allocating fictive shared memory arrays, that limits the maximum number of active thread blocks per multiprocessor. Assuming that ϱ is limited by shared memory, from (1) and (3) we got the occupancy value $\varrho = N_t \left[\frac{N_{smax}}{N_s}\right]/N_{tmax}$. Thus by allocating different-size shared memory arrays we can control ϱ by choosing it from the range $\varrho = N_t/N_{tmax},\ 2N_t/N_{tmax},\ \ldots,\ 1$. Using such procedure the function $W(\varrho)$ was determined from model (12) for different ρ. The obtained results are presented in Fig. 1.

In order to analyze the dependence of β_2 on sizes of blocks of threads in x and y directions d_x, d_y, we have computed the test problem with different values

Table 2. Computation times with different sizes of thread blocks on GPU2

d_x	d_y =32	d_y = 64	d_y = 128
1	—	0.090342	0.0883086
2	0.0902548	0.0857643	0.0838882
4	0.0914037	0.0876535	—
8	0.101497	—	—

of these parameters. As a benchmark for each iteration we transfer three arrays with halo elements from the global to the shared memory and copy one simple array back to the global memory. Results are presented in Table 2.

It follwos that the computation times are roughly the same for different sizes of block of threads. However, this conclusion does not apply to GPU3 card – in this case β_h must be determined for each fixed block size. For GPU3 the smaller block sizes give better results, our guess is that GPU3 manages to synchronize smaller blocks better, since there are less threads for synchronization.

Thus we get from our theoretical performance model that $\beta_h = \frac{TW(\varrho)-N}{d_h N}$. We present values of parameters $(\beta_0, \beta_h, \gamma)$ for different GPUs, here $\beta_0 = 1/W(1/3)$: $(4.22E{-}10, 7.5, 19.4E{-}12)$ for GPU1, $(4.92E{-}11, 2.15, 1.60E{-}12)$ for GPU2, $(6.70E{-}11, --, 0.92E{-}12)$ for GPU3. The value β_h for GPU3 depends on block sizes, however it is similar to GPU2 value. So if β has much larger impact on calculation time than γ, then the complexity model of GPU2 predicts the smallest computation times.

Next we consider the numerical scheme which was used to solve 3D electromagnetic problem that describes a wave propagation in a rectangular waveguide. The details of the algorithm are presented in [1]. The main part of one time step calculations can be represented in the following form

$$\begin{aligned}
H_{x,ijk} &+= s_z(E_{y,i,j,k+1} - E_{y,i,j,k}) - s_y(E_{z,i,j+1,k} - E_{z,i,j,k}),\\
H_{y,ijk} &+= s_x(E_{z,i+1,j,k} - E_{z,i,j,k}) - s_z(E_{x,i,j,k+1} - E_{x,i,j,k}),\\
H_{z,ijk} &+= s_y(E_{x,i,j+1,k} - E_{x,i,j,k}) - s_x(E_{y,i+1,j,k} - E_{y,i,j,k}),\\
E_{x,ijk} &+= s_y(H_{z,i,j,k} - H_{z,i,j-1,k}) - s_z(H_{y,i,j,k} - H_{y,i,j,k-1}),\\
E_{y,ijk} &+= s_z(H_{x,i,j,k} - H_{x,i,j,k-1}) - s_x(H_{z,i,j,k} - H_{z,i-1,j,k}),\\
E_{z,ijk} &+= s_x(H_{y,i,j,k} - H_{y,i-1,j,k}) - s_y(H_{x,i,j,k} - H_{x,i,j-1,k}),
\end{aligned}$$

where s_x, s_y, s_z are constants. In the performance prediction model we used the following parameters of GPU2 card: $d_h = 1/3$, $\beta_h = 3.27$, $W(2/3) = 9.46E - 10$, $\gamma = 1.72E - 12$, $N = 256^3 \cdot 10 \cdot 12$, $\bar{N} = 256^3 \cdot 10 \cdot 36$. Then we have predicted computation time

$$T = (1 + \beta_h d_h)N/W(\varrho) + \gamma \bar{N} = 0.278\,\mathrm{s}. \tag{13}$$

The real calculations took 0.177 s. So our model predicts correctly the main order of computation time.

Table 3. The comparison of shared memory and L1 cache usage for different stencils

Method	3 of 7	7 of 7	27 of 27
L1 cache	0.179	0.266	0.711
Shared memory, cross stencil	0.177	0.186	–
Shared memory, box stencil	0.227	0.226	0.231

Video cards of compute capability 2 and higher support data L1 caching technology, that acts similarly to the shared memory, but the optimization is done automatically. Can it replace the shared memory? The answer is positive at least in some cases. We have done computational experiments with three different patterns of stencils. First, we have used the standard cross stencil, when seven points define the set of neighbours. In this case we have analyzed two scenarios, when only three points of the set are used and when all seven points are used. Second, we applied the full 3D box stencil with 27 neighbours. Note, that this stencil enables also the implementation of algorithms requiring smaller stencils (such as cross stencil). In Table 3 the results of experiments are presented, where different stencils are used. The cases of shared memory and L1 cache usage are compared.

From the presented results we see that the box-type stencil for shared memory implementation is an efficient and robust alternative to specific types of stencils.

References

1. Kancleris, Ž., Šlekas, G., Čiegis, R.: Sensitivity of asymmetrically necked planar structure for microwave pulse power measurement in rectangular waveguide. IEEE Sensors J. **13**(4), 1143–1147 (2013)
2. Adams, S., Payne, J., Boppana, R.: Finite difference time domain (FDTD) simulations using graphics processors. In: DoD High Performance Computing Modernization Program Users Group Conference, pp. 334–338 (2007)
3. Liuge, D., Kang, L., Fanmin, K.,: Parallel 3D finite difference time domain simulations on graphics processors with Cuda. Comput. Intell. Softw. Eng. pp. 1–4 (2009)
4. Micikevicius, P.: 3D finite difference computation on GPUs using CUDA. In: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, pp. 79–84 (2009)
5. Starikovičius, V., Čiegis, R., Iliev, O.: A parallel solver for optimization of oil filters. Math. Model. Anal. **16**(2), 326–342 (2011)

OpenACC Parallelisation for Diffusion Problems, Applied to Temperature Distribution on a Honeycomb Around the Bee Brood: A Worked Example Using BiCGSTAB

Hermann J. Eberl(✉) and Rangarajan Sudarsan

Department of Mathematics and Statistics, University of Guelph, Guelph, ON, N1G2W1, Canada
{heberl,rsudarsa}@uoguelph.ca

Abstract. We discuss a simple OpenACC implementation of the iterative BiCGSTAB algorithm for linear systems. Problems of this type arise in the numerical solution of diffusion-reaction problems, where the linear solver constitutes the most computationally expensive component of the simulation (∼80 % of time spent) and therefore has often been the primary target for parallelization. We deploy and test this method on a desktop workstation with two supported GPUs, one targeted for high performance computing, one a consumer level GPU, to compute the temperature distribution on a honeycomb around the bee brood. The paper is written from a user's, not from a GPU computing expert's perspective and aims to fill a gap we noticed between real world application problems and the simple problems solved in introductory OpenACC tutorials or in benchmarking studies.

Keywords: Diffusion equation · GPU · OpenACC · Krylov-subspace methods · Honeycomb · Bee brood

1 Introduction

With the availability of multicore processors, multi-processoraut]Eberl, Hermann J. aut]Sudarsan, Rangarajan desktop computers, and more recently also general purpose computing graphical processing units, parallel computing has entered mainstream in computational science [3,5]. In the past, parallel computers were tools available to only select few researchers working on the computationally most demanding problems, who used their years of training to optimally utilize the parallel computing resources. With the current availability, every computational scientist has the opportunity to use them and has to face the steep learning curve and challenges that come with parallel computing, in order to take advantage of recent advances even in desktop computing technology. OpenACC is a programming standard for GPU parallelization that is based on extensions of the programing languages C and Fortran [8,10]. As in OpenMP for shared memory multi-processor/multi-core platforms, the user inserts directives in his code to explicitly instruct the compiler to parallelize sections on a supported GPU accelerator device. While OpenACC code is not as highly performing as hand tuned, optimized CUDA code, it allows quick access to GPU computing without major re-programming of existing code [2,15]. Therefore, it should be attractive for occasional GPU computing users, and for scientists whose primary interest is not to produce speed optimized production code, but to develop prototype or research codes. Moreover, OpenACC codes have the advantage that compilers that do not recognize its directives, ignore them without crying foul and produce traditional CPU code instead. Several papers with benchmark tests of OpenACC and comparisons against other approaches for such benchmark problems can be found in the literature, e.g. [11], but applications to real world problems are only slowly to emerge [2,15].

In this paper we want to investigate how one can utilize OpenACC on a desktop workstation for the numerical solution of diffusion-reaction equations. Problems of this type arise in many areas of computational science and engineering, such as mass and heat transfer studies, or wave propagation phenomena in chemical engineering or mathematical biology and ecology. In diffusion problems, the computationally most expensive part, and therefore the primary target for parallelisation, is the solution of large, sparse, structured (often banded) linear algebraic systems [1,7]. As an example we use the stabilised biconjugate gradient method BiCGSTAB [12] for this task. We describe its parallelization with OpenACC and, for illustration purposes, use it to compute the temperature distribution on a honeycomb around the bee brood in a Langstroth beehive on a workstation with two supported GPUs, one specific for high performance computing, one a consumer level GPU that can be found in regular desktop PCs.

The paper is written from a user's, not from a GPU computing expert's perspective. It tries to fill a gap that we feel exists between the research literature and introductory OpenACC tutorials. The reader can easily and quickly find tutorials on the internet, which often focus on the most simple applications of parallelization directives, such as the parallelization of single loop nests with constant scalar coefficients. Some, e.g. [9,13], discuss the Jacobi method for the solution of diffusion problems, a basic linear iteration that lends itself directly and in a straight forward manner to loop level parallelization. For real world problems, however, one would resort to more involved but faster methods, such BiCGSTAB [12]. We want to show how methods of this class can be parallelised in OpenACC.

2 Mathematical Model

2.1 General Setting

Many applications in science an engineering involve the numerical solution of systems of diffusion-advection-reaction equations of the form

$$u_t = \nabla\left(D(t,x,u)\nabla u\right) - \nabla(v(t,x)u) + f(t,x,u), \quad x \in \Omega \subset \mathbb{R}^d, t \geq 0 \qquad (1)$$

where $d \in \{2, 3\}$. Here, the dependent variable is the r-dimensional vector valued function $u(t, x)$; the $r \times r$ matrix valued function $D(t, x, u)$ contains the self- and cross-diffusion coefficients, $v(t, x)$ denotes the underlying convective velocity field, and the r-dimensional vector valued function $f(t, x, u)$ subsumes reactions, sinks, and sources; for specific applications (1) needs to be augmented with appropriate consistent initial and boundary conditions. Characteristic for most problems of this type is that time-explicit methods usually are subject to heavy time-step constraints, which mandate the use of implicit or semi-implicit methods. After discretization, in every time step, one or more algebraic systems must be solved. Depending on D and f, and on the time integration method used (full implicit vs. linearly implicit vs. semi-implicit) these algebraic systems can be nonlinear or linear. Nonlinear systems are typically reduced to solving a family of linear problems, e.g. by Newton-like methods or fix-point iterations.

If one considers the sometimes simpler special case of a time-independent or steady-state problem

$$0 = \nabla \left(D(x, u)\nabla u\right) + f(x, u), \quad x \in \Omega \subset \mathbb{R}^d \tag{2}$$

then spatial discretization leads directly to such a linear or nonlinear algebraic system, depending on D, f. In any case, the computational core problem of solving diffusion-reaction problems of type (1) or (2) is to solve one or many linear systems of the form

$$Ax = b, \quad A \in \mathbb{R}^{N \times N}, \quad x, b \in \mathbb{R}^N, \tag{3}$$

where matrix A contains the discretisation of the (linearized) diffusion-reaction operator, vector b contains sinks/sources, additional reaction terms, contributions of boundary conditions, and in time-dependent problems information from the previous time level. Vector x is the numerical approximation of the solution u on the computational grid. N is the number of grid points.

For common spatial discretisation schemes, such as standard Finite Difference, Finite Element, or Finite Volume methods, the system matrices A are sparse, often structurally symmetric (e.g. banded) but typically not symmetric if D is not diagonal or $v \not\equiv 0$. In many cases, the matrix can be represented in sparse diagonal format.

2.2 OpenACC Parallelization of the Linear Solver

We focus on the parallelization of the linear solver for (3), which is the computationally most demanding part of the solution of diffusion problems. A popular class of numerical methods for such problems are iterative Krylov subspace methods [12], and among those BiCGSTAB is often the method of choice for non-symmetric problems, whereas the Conjugate Gradient (CG) method is typically preferred for symmetric problems. Krylov subspace methods require per iteration step only few matrix-vector products (two in the case of BiCGSTAB, one for CG), few inner products and several vector additions and multiplications of vectors with scalars. These scalars are not constants but change dynamically from iteration to iteration, as the method converges to the solution.

We discuss here in detail the more widely applicable BiCGSTAB algorithm. The basic algorithm has been extensively described in the literature [12]. The simpler CG method can be treated similarly. In the next section we will present results for both methods.

When parallelizing the method, the approach is to parallelize the vector operations and the matrix-vector product. In OpenACC this is accomplished by inserting the loop-directive. A crucial aspect in parallel computing is the trade-off between maximizing parallel execution of the work to be done and minimizing communication between nodes. In GPU computing, this means to minimize the communication between the GPU device and the host device. The most efficient GPU algorithms are those that live entirely on the GPU without accessing host memory or invoking host CPU computations. Therefore, we implement the method such that communication between device and host is only required in the beginning (to hand over data from the host to the device) and at the end (to hand back the result) of the BiCGSTAB method. To this end we define a single OpenACC data region inside which all calculations take place. The auxiliary vector and scalar variables used by the algorithm are created directly in the GPU device memory and need not to be communicated with the host. As a consequence, all parallel kernels (including for the computation of the matrix-vector and inner products) can use existing data without communicating with the host. Pseudocode of the parallelized BiCGSTAB algorithm, indicating where OpenACC directives are used, is given in Algorithm 1. This algorithms calls for the evaluation of matrix-vector products in lines 5, 19, 26 and inner products in lines 13, 20, 27, 28. Fortran stubs of a subroutine for the matrix vector product with sparse matrix in diagonal format and a function for the inner products can be found in Appendix A. In applications, in which the sparse matrix is stored in a different format, the former has to be replaced accordingly. At present, OpenACC does not support Fortran array assignments and built in array functions, such as `dot_product`. Therefore, the latter has do be provided. Both, matrix-vector product subroutine and inner product function, assume that they are called from within a parallel data region. Stopping criteria for the iterative method are implemented as reduction operations in the last parallel loop region, starting at line 30.

3 Example: Temperature on a Honey Bee Comb

3.1 Problem Description

As a test we study a scalar stationary linear problem of type (1), which requires only a single solution of a linear system: the steady state temperature distribution around the bee brood in a honeycomb frame of a Langstroth hive, as introduced in [4], see also [14]. Honeybee colonies must keep the temperature around their brood in a narrow band of few degrees. Temperatures below or above will lead to malformations or death of the brood. The content of the

```
**Data**: matrix $A$, right hand side $b$; initial guess for solution $x$
**Result**: solution $x$
OpenACC Data Region
    copyin: $A, b$;
    copy: $x$;
    create: $v, p, r, s, t, u$;
    $r_0 := Ax$ ;
    OpenACC Parallel Loop
        present: $r_0, b, v, p, r$;
        for $j = 1, ..., n$ do
            $r_{0,j} := b_j - r_{0,j}$;
            $v_j := p_j := 0, r_j := r_{0,j}$;
    $\rho := \alpha = \omega_0 := 1$;
    for $i = 1, 2, ...$ do
        $\rho_i := (r_0, r)$ ;
        $\beta := (\rho_i/\rho_{i-1})(\alpha/\omega)$;
        OpenACC Parallel Loop
            present: $p, r, v$; firstprivate: $\beta, \omega$;
            for $j = 1, ..., n$ do
                $p_j := r_j + \beta(p_j - \omega v_j)$;
        $v := Ap$ ;
        $\delta_1 := (r_0, v)$ ;
        $\alpha := \rho_i/\delta_1$;
        OpenACC Parallel Loop
            present: $s, r, v$; firstprivate: $\alpha$;
            for $j = 1, ..., n$ do
                $s_j := r_j - \alpha v_j$;
        $t := As$ ;
        $\delta_2 := (t, s)$;
        $\delta_3 := (t, t)$;
        $\omega := \delta_2/\delta_3$ ;
        OpenACC Parallel Loop
            present: $p, s, x, r, t, u$; firstprivate: $\alpha, \omega$;
            for $j = 1, ..., n$ do
                $u_j := \alpha p_j + \omega s_j$;
                $x_j := x_j + u_j$;
                $r_j := s_j - \omega t_j$;
```

Algorithm 1. BiCGSTAB algorithm for the solution of the linear algebraic system $Ax = b$ with $A \in \mathbb{R}^{N\times N}, b, x \in \mathbb{R}^N$. Lower case Latin letters are vectors in $\mathbb{R}^n$ (with the exception of integer counters i, j), Greeks are scalars. OpenACC blocks are indicated by **OpenACC**, OpenACC clauses are bold faced. OpenACC currently does not support Fortran array assignments, hence vector operations must be implemented as explicit loops, as indicated here.

Table 1. Model parameters used in the simulation of (4)

Parameter	Symbol	Unit	Air	Honey	Pollen	Bees	Pupae
Conductivityheat generation	k	$Wm^{-1}K^{-1}$	0.02624	0.60	0.15	0.61	0.61
	G	Wm^{-3}	0	0	0	$2.9 \cdot 10^4$	10^3

individual comb cells varies across the comb. Cells can be occupied by honey, pollen, pupae, bees, or be unoccupied (air), see also Fig. 1. This leads to a linear, non-homogeneous diffusion problem with spatially varying coefficients,

$$0 = \nabla\left(k(x)\nabla u\right) + G(x). \tag{4}$$

on a rectangular domain $\Omega = L \times H$. We use $H = 0.08\,m, L = 0.1\,m$. Distributed parameter k is the local thermal conductivity and G is the heat generation rate. Both depend on the local content of the honey bee comb and are summarized in Table 1. Following [4], we prescribe at the boundary the temperature as $u = 34^oC$. The steady state temperature distribution according to this model provides a reference for comparison of the efficiency of so-called heater bees, i.e. worker bees who actively engage in producing heat in order to keep the brood warm, which is described by a non-autonomous transient version of (4), see [4].

We use a standard 2nd order Finite Difference based Finite Volume method on a uniform grid of size $n \times m$ [6], with $\Delta x = \Delta y$, and $m := \frac{H}{L}n$. The dependent variable u is computed in the centers of the grid cells. The material properties heat conductivity and heat generation are also given in the centers of the grid cell. The heat fluxes $J := -k(x)\nabla u$ must be evaluated at the faces of the grid cells. We obtain them as the harmonic average from the neighbouring grid cells. The flux at the interface between grid cells (i, j) and $(i + 1, j)$ is given by

$$J_{i+1/2,j} = -\frac{2k_{i+1,j}k_{i,j}}{k_{i+1,j} + k_{i,j}} \cdot \frac{u_{i+1,j} - u_{i,j}}{\Delta x},$$

where $u_{i,j}, k_{i,j}$ denote u and k in the center of cell (i, j). For the heat fluxes across the remaining edges of cell (i, j) we have accordingly

$$\begin{aligned}
J_{i-1/2,j} &= -\frac{2k_{i,j}k_{i-1,j}}{k_{i,j} + k_{i-1,j}} \cdot \frac{u_{i,j} - u_{i-1,j}}{\Delta x},\\
J_{i,j+1/2} &= -\frac{2k_{i,j}k_{i,j+1}}{k_{i,j} + k_{i,j+1}} \cdot \frac{u_{i,j+1} - u_{i,j}}{\Delta x},\\
J_{i,j-1/2} &= -\frac{2k_{i,j}k_{i,j-1}}{k_{i,j} + k_{i,j-1}} \cdot \frac{u_{i,j} - u_{i,j-1}}{\Delta x}.
\end{aligned}$$

The coefficients of matrix A are then obtained from the linear equations for $u_{i,j}$

$$\frac{1}{\Delta x}\left(J_{i+1/2,j} - J_{i-1/2,j}\right) + \frac{1}{\Delta x}\left(J_{i,j+1/2} - J_{i,j-1/2}\right) = G_{i,j}$$

In boundary cells this expression accesses points outside the domain which are eliminated in the usual way by replacing the grid external values with the boundary data, which leads to additional contributions to the right hand side in these cells. In order to be able to use our linear solver, we convert the problem from grid notation to vector notation using the lexicographical grid ordering

$$\pi : \{1, ..., n\} \times \{1, ..., m\} \to \{1, ..., nm\}, (i, j) \mapsto (i-1)m + j$$

such that $x_{\pi(i,j)} := u_{i,j}$, etc. The resulting linear system takes then the form

$$Ax = b, \quad A \in \mathbb{R}^{nm \times nm}, \quad x, b \in \mathbb{R}^{nm}.$$

Model (4) is a linear scalar problem without convective terms. This is the simplest problem of type (1). Matrix A is symmetric and in a real application study would be tackled more efficiently with the Conjugate Gradient method (CG). We present below results obtained with both methods.

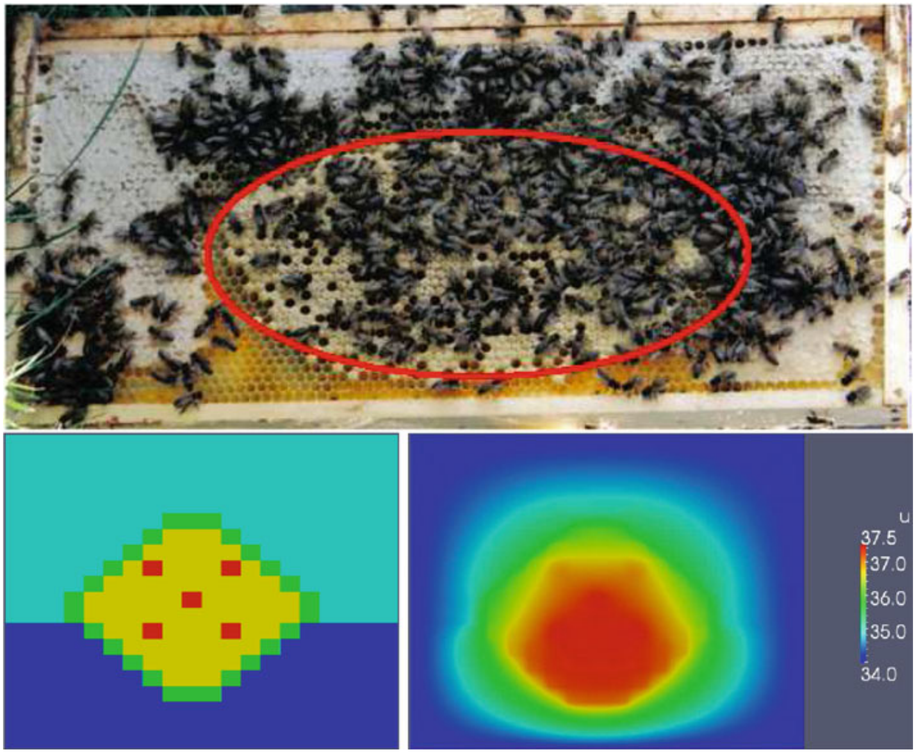


Fig. 1. top: frame of a honeybee hive showing the heterogeneity of cell contents (courtesy Cody Thompson); **bottom left:** Distribution of cell content across the honey bee comb in the computational model: air (dark blue), honey (light blue), pollen (green), pupae (yellow), bees (red); **bottom right:** temperature distribution u according to (4) (Color figure online).

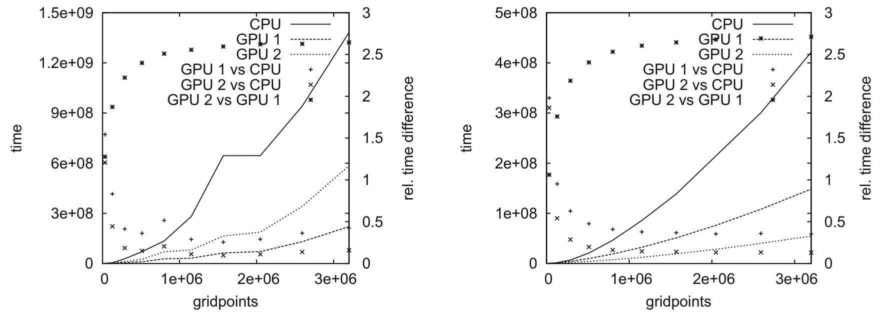


Fig. 2. Results for the test problem for BiCGSTAB (left) and the Conjugate Gradient method (right): Compute time, as reported by Fortran subroutine `system_clock`, in dependence of grid size. Shown are absolute values (lines) and execution times relative to each other (symbols). GPU1: Tesla K20c, GPU2: GeForce GTX650. Slightly different stopping criteria were used for both methods, therefore execution times should not be directly compared.

3.2 Results

The tests were run on a custom built workstation with an Intel Core i7-3970X Extreme CPU (3.5 GHZ, 15 MB cache) and 16 GB RAM under Ubuntu 12.04. It has two OpenACC supported NVIDIA GPUs, a Tesla K20c (5 GB global memory, 13 multiprocessors with 2496 cores) and a consumer level GeForce GTX 650 (1 GB, 2 multiprocessors, 384 cores). The Portland Group Fortran compiler version 13.3.0 was used; optimization was restricted to the compiler options that are recommended to work well in most cases. We solve (4) for grid resolutions $n \times m = 0.8n^2$ for $n = 200, ..., 2000$, and record the computation time required by the linear solver on CPU and GPUs.

In Fig. 1 we show a photograph of a brood comb and the distribution of comb content in the computational model, along with the computed temperature. The bottom part of the comb is empty (air), the top part of the comb is filled with honey. Brood (pupae) is found in the center, surrounded by cells with pollen. Interspersed in the brood are pockets occupied with heater bees. The temperature is highest where the bees are and decreases toward the boundary of the domain. The brood is contained in a region with temperature between approx. 36 °C and 37.5 °C. Although the distribution of the brood in the honeycomb is symmetric in y-direction, the resulting temperature distribution is not, due to spatially heterogeneous heat conductivity.

In Fig. 2 we plot the time used for the solution of the linear system on the CPU and both GPUs, for the OpenACC BiCGSTAB algorithm described in Algorithm 1 and for a corresponding OpenACC implementation of the conjugate gradient method (implementation not discussed here due to space restrictions). The performance of both methods is qualitatively similar. Computing time increases with gridsize for all three types of hardware. For the smallest grid resolution the CPU computations are faster than the GPUs. As the grid increases, the Tesla K20 performs best. The relative computing time between GPU and CPU levels off at a ratio of approximately 0.16:1 (BiCGSTAB) or 0.13:1 (CG). For the consumer level GPU GeForce GTX650 we observe a ratio of 0.42:1 (BiCGSTAB) and 0.35:1 (CG). The ratio of compute times between both GPUs for our problem levels off at approx. 2.7:1 for both methods. We observe a clear acceleration of the computation on the GPUs compared to the CPU, provided the grid is large enough so that communication between host and device becomes small compared to the time needed for the actual computations. In the case of BiCGSTAB this is achieved by inserting four explicit loop parallelization directives in the linear solver and one each in the matrix-vector product and the inner-product.

4 Conclusion

The numerical simulation of diffusion-reaction problems in science and engineering involves the solution of structured, large, sparse linear systems. This often is the computationally most expensive part and the primary target for parallelization. With GPU computing becoming widely accessible, even on desktops and workstations, directive based GPU programming language extensions become attractive for many users who want to benefit from the hardware quickly, without extensive re-programming of their codes. We demonstrate here an OpenACC parallelization of BiCGSTAB, a Krylov subspace linear solver, that gives quick access to the GPU compute capabilities. For sufficiently large problems the OpenACC parallelized GPU version clearly outperforms the CPU version, not only on the HPC GPU system, but also on the OpenACC supported consumer level GPU.

Acknowledgement. Hardware and compiler (excluding the Tesla K20c) were purchased with a NSERC-RTI Grant. The TESLA K20c was donated by NVIDIA. We thank Larry Banks for the setup of the equipment and two anonymous referees for useful suggestions.

Appendix

A OpenACC Fortran Code-stubs

The following code stubs are provided for illustration only. They are copied from the source codes used here but changed to reflect the notation in the text. The authors do not guarantee their proper working if copied from here and do not assume responsibility for any damages this might cause.

Matrix-vector product $y := Ax$ for sparse diagonal format. In the following code, d is the number of diagonals, sparse matrix A is an $n \times d$ array, *ioff* contains the offsets of the sub-diagonals. $A, x, d, ioff$ are handed over to the subroutine, y is returned.

```
do j=1, d
   io = ioff(j)
   i1 = max0(1,1-io)
   i2 = min0(n,n-io)

   !$acc parallel loop copyout(y) &
   !$acc present(x,diag) firstprivate(io)
   do i=i1,i2
      y(i) = y(i)+A(i,j)*x(i+io)
   enddo
   !$acc end parallel loop
enddo
```

Scalar product *ddt* $:= x^T y$. In the following code stub x, y are n-arrays handed over to the function, the real variable *ddt* is returned.

```
!$acc kernels loop private(ddt) reduction(+: ddt)
do i=1,n
   ddt=ddt+x(i)*y(i)
enddo
```

References

1. Čiegis, R., Čiegis, R., Jakušev, A., Šaltenienė, G.: Parallel variational iterative algorithms for solution of linear systems. Math. Mod. An. **12**, 1–16 (2007)
2. Herdman, J.A., Gaudin, W.P., McIntosh-Smith, S., Boulton, M., Beckingsdale, D.A., Mallinston, A.C., Jarvis, S.A.: Accelerating hydrocodes with OpenACC, OpenCL and CUDA. In: Proceedings of the SC Companion: High Performance Computing Networking Storage and Analysis, pp. 465–471 (2012)
3. Herlihy, M., Luchangco, V.: Distributed computing and the multicore revolution. ACM SIGCAT News **39**, 62–72 (2008)
4. Humphrey, J.A.C., Dykes, E.S.: Thermal energy conduction in a honey bee comb due to cell-heating bees. J. Theor. Biol. **250**, 194–208 (2008)
5. Marowka, A.: Parallel computing on any desktop. Commun. ACM **50**, 75–78 (2007)
6. Morton, K.W.: Numerical solution of convection-diffusion problems. Chapman and Hall, London (1996)
7. Muhammad, N., Eberl, H.J.: OpenMP parallelization of a mickens time-integration scheme for a mixed-culture biofilm model and its performance on multi-core and multi-processor computers. In: Mewhort, D.J.K., Cann, N.M., Slater, G.W., Naughton, T.J. (eds.) HPCS 2009. LNCS, vol. 5976, pp. 180–195. Springer, Heidelberg (2010)
8. The OpenACC Application Programming Interface Version 1.0. http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf
9. Poole, D.: Introduction to OpenACC Directives. http://on-demand.gputechconf.com/gtc/2012/presentations/S0517A-Monday-Programming-GPUs-OpenACC.pdf
10. The Portland Group, PGI Accelerator Compilers OpenACC Getting Started Guide, Version 13.2. http://www.pgroup.com/doc/openACC_gs.pdf

11. Reyes, R., López, I., Fumero, J.J., de Sande, F.: Directive based programming for GPUs: a comparative study. In: Proceedings of the IEEE 14th International Conference on HPCC, pp. 410–417 (2012)
12. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. SIAM, Philadelphia (2003)
13. Strelchenko, A.: Parallel programming with OpenACC directives. http://linksceem.eu/ls2/images/stories/openacc.pdf
14. Sudarsan, R., Thompson, C.G., Kevan, P.G., Eberl, H.J.: Flow currents and ventilation in Langstroth beehives due to brood thermoregulation efforts of honeybees. J. Theor. Biol. **295**, 168–193 (2012)
15. Wienke, S., Springer, P., Terboven, C., an Mey, D.: OpenACC – first experiences with real-world applications. In: Kaklamanis, C., Papatheodorou, T., Spirakis, P.G. (eds.) Euro-Par 2012. LNCS, vol. 7484, pp. 859–870. Springer, Heidelberg (2012)

Application of CUDA for Acceleration of Calculations in Boundary Value Problems Solving Using PIES

Andrzej Kuzelewski, Eugeniusz Zieniuk, and Agnieszka Boltuc(✉)

Institute of Computer Science, University of Bialystok,
Sosnowa 64, 15-887 Bialystok, Poland
{akuzel,ezieniuk,aboltuc}@ii.uwb.edu.pl
http://ii.uwb.edu.pl

Abstract. The main purpose of this paper is examination of an application of modern parallel computing solutions to speed up the calculation in the numerical solution of parametric integral equations systems (PIES). Solving boundary value problems by PIES sometimes requires large computing time, particularly in more complex problems. This paper presents use of graphics cards programming in general-purpose applications (GPGPU). The boundary value problems modelled by 3D Navier-Lamé equations are solved using PIES and NVidia CUDA technology. The testing example shows that the use of GPGPU significantly increases speed of calculations in PIES.

Keywords: Parametric integral equations systems · CUDA · GPGPU · Boundary value problems

1 Introduction

For many years the authors of this paper have worked on developing and application of parametric integral equations systems (PIES) to solve boundary value problems. PIES has been already used to solve problems modelled by 2D and 3D partial differential equations, such as: Laplace, Poisson, Helmholtz and Navier-Lamé [7–10,12,14]. These equations were solved numerically using PIES. This method includes a shape of a boundary of considered problem in its mathematical formalism. In order to include a shape of a boundary, the curves (eg. Bézier, B-spline and others) and the surfaces (such as Coons, Bézier and others) known from computer graphics was applied in PIES. It results a small number of control points required to define the shape of a boundary.

Former studies focused on accuracy and efficiency of the results obtained using PIES in comparison with well-known algorithms such as FEM or BEM as well as analytical methods. The authors have taken into consideration more sophisticated problems such as identification of boundary geometry [11,13], as well. These studies confirmed the effectiveness of PIES, however analyzed more complex problems forced the authors to pay attention to computing time, as well. In general, the greater number of input data, the longer an algorithm works. It was noticed in the case of some more complex problems, which were modelled by 3D Navier-Lamé equations and solved using PIES, as well.

Increasing speed of the algorithm can be achieved in the following ways: by optimization of the existing code as well as use more advanced computers, multiprocessor machines, clusters or GPUs. Recently, researchers increased their focus on the implementation of graphics cards (GPUs) for numerical calculations in general-purpose applications (GPGPU), due to great possibility to improve computing performance [3,5,6]. It is related to the architecture of graphics cards (multi-processor and multi-threaded), very fast floating-point arithmetic units and use of high-speed memory. Therefore, the authors examined an application of modern parallel computing solutions to increase efficiency of calculations in the numerical solution of PIES. It was decided to use Compute Unified Device Architecture (CUDA) - parallel computing platform and programming model invented by NVidia [1].

This paper presents a possibility of acceleration numerical calculations in PIES using CUDA technology. The boundary value problems modelled by 3D Navier-Lamé equations were considered.

2 PIES for 3D Navier-Lamé Equations

PIES for three-dimensional Navier-Lamé equations were obtained as the result of analytical modification of boundary integral equation (BIE). A detailed description of the methodology of modification for 2D problems modelled by various differential equations is presented in [7–10]. Generalization of mentioned methodology to 3D problems results the following formula of PIES [8]:

$$\frac{1}{2}u_l(v_1,w_1) = \sum_{j=1}^{n} \int\limits_{v_{j-1}}^{v_j} \int\limits_{w_{j-1}}^{w_j} \{\overline{\mathbf{U}}_{lj}(v_1,w_1,v,w)p_j(v,w) - \overline{\mathbf{P}}_{lj}(v_1,w_1,v,w)u_j(v,w)\}J_j(v,w)dvdw \tag{1}$$

where function $Jj(v,w)$ is the Jacobian, $v_{l-1} < v_1 < v_l$, $w_{l-1} < w_1 < w_l$, $v_{j-1} < v < v_j$, $w_{j-1} < w < w_j$, $\{l,j\} = 1,2,...,n$, whilst n - is the number of parametric patches that create the domain boundary in 3D.

Integrands $\overline{\mathbf{U}}_{lj}(v_1,w_1,v,w)$, $\overline{\mathbf{P}}_{lj}(v_1,w_1,v,w)$ in (1) are presented in the following matrix form [8]:

$$\overline{\mathbf{U}}^{*}_{lj}(v_1,w_1,v,w) = \frac{1}{16\pi(1-\nu)\mu\eta}\begin{bmatrix} U_{11} & U_{12} & U_{13} \\ U_{21} & U_{22} & U_{23} \\ U_{31} & U_{32} & U_{33} \end{bmatrix}, \quad \mu = \frac{E}{2(1+\nu)}, \tag{2}$$

$$\overline{\mathbf{P}}^{*}_{lj}(v_1,w_1,v,w) = \frac{1}{8\pi(1-\nu)\eta^2}\begin{bmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{bmatrix}. \tag{3}$$

The individual elements in the matrices (2) and (3) in the explicit form can be found in [8]. Integrands (2) and (3) include in its mathematical formalism the shape of a closed boundary surface. It is created using appropriate relationships between the patches $(l, j = 1, 2, ..., n)$, which are defined in Cartesian coordinates as follows [8]:

$$\begin{aligned} \eta_1 =& P_j^{(1)}(v,w) - P_l^{(1)}(v_1,w_1),\ \eta_2 = P_j^{(2)}(v,w) - P_l^{(2)}(v_1,w_1),\\ &\eta_3 = P_j^{(3)}(v,w) - P_l^{(3)}(v_1,w_1),\ \eta = \sqrt{\eta_1^2+\eta_2^2+\eta_3^2}, \end{aligned} \tag{4}$$

where $P_j^{(1)}$, $P_j^{(2)}$, $P_j^{(3)}$ are the scalar components of the vector surface $\mathbf{P}_j(v,w) = \left[P_j^{(1)}(v,w), P_j^{(2)}(v,w), P_j^{(3)}(v,w)\right]^T$, which depends on v and w parameters. This notation is also valid for the surface marked by l with parameters v_1, w_1, i.e. for $j = l$ and parameters $v = v_1$ and $w = w_1$. Parameters ν and E in (2) and (3) are material constants: Poisson's ratio and Young modulus respectively.

In 3D problems modelled using PIES, vector functions $\mathbf{P}_j(v,w)$ have the form of parametric surface patches widely applied in computer graphics. The main advantage of presented approach, compared with BIE, is analytical inclusion of the boundary directly in PIES. The boundary is not included in mathematical formalism of BIE, however is defined in general way by the boundary integral. Therefore, discretization of the boundary into elements is required, as is the case of BEM. The advantages of including the boundary directly in mathematical equations (PIES) were widely presented in 2D [7–10] and 3D [8,12,14] problems.

The application of PIES for solving 2D and 3D problems allows to eliminate discretization of the boundary, as well as the boundary functions. In previous studies, the boundary functions both defined as boundary conditions, as well as obtained after solving PIES are approximated by Chebyshev polynomials [7,9,10]. Unlike the problems modelled by Laplace's equation, ones described by the Navier-Lamé equations require boundary functions in vector form. Therefore, it were generalized previously used approximating series. They represent scalar components of vectors of displacements $\mathbf{u}_j(v,w)$ and stresses $\mathbf{p}_j(v,w)$. Boundary functions on each j surface are approximated by the following series:

$$\mathbf{u}_j(v,w) = \sum_{p=0}^{N}\sum_{r=0}^{M} \mathbf{u}_j^{(pr)} L_j^{(p)}(v) L_j^{(r)}(w), \tag{5}$$

$$\mathbf{p}_j(v,w) = \sum_{p=0}^{N}\sum_{r=0}^{M} \mathbf{p}_j^{(pr)} L_j^{(p)}(v) L_j^{(r)}(w), \tag{6}$$

where $\mathbf{u}_j^{(pr)}$, $\mathbf{p}_j^{(pr)}$ are unknown coefficients, and $L_j^{(p)}(v)$, $L_j^{(r)}(w)$ have the following form:

$$L_j^{(k)}(x) = \frac{(x-x_0)(x-x_1)...(x-x_{i-1})(x-x_{i+1})...(x-x_l)}{(x_j-x_0)(x_j-x_1)...(x_j-x_{i-1})(x_j-x_{i+1})...(x_j-x_l)}, \tag{7}$$

where: $k = \{p, r\}$, $l = \{M, N\}$,$x = \{v, w\}$.

After substituting (5) and (6) to (1) and writing down at proper collocation points [2] we obtain a system of linear algebraic equations with respect to unknown coefficients. Coefficients $u_j^{(pr)}$ or $p_j^{(pr)}$ can be obtained by solving the mentioned system [8]. It should be noticed that always one of these coefficients on the individual segments are derived from boundary conditions by approximating series (5) or (6).

After solving PIES only solutions on the boundary are obtained. They are represented by approximation series (5) or (6). In order to obtain solutions in the domain, analytical modification of integral identity from BIE is required. New integral identity is formulated in the same way as in 2D problems [7,8]. The identity uses the results which were previously obtained by solving PIES and it take the following form [8]:

$$\mathbf{u}(\mathbf{x}) = \sum_{j=1}^{n} \int_{v_{j-1}}^{v_j} \int_{w_{j-1}}^{w_j} \{\widehat{\overline{\mathbf{U}}}_j(\mathbf{x}, v, w) p_j(v, w) - \widehat{\overline{\mathbf{P}}}_j(\mathbf{x}, v, w) u_j(v, w)\} J_j(v, w) dv dw \tag{8}$$

Integrands in identity (8) are presented in the following form [8]:

$$\widehat{\overline{\mathbf{U}}}_j^*(\mathbf{x}, v, w) = \frac{1}{16\pi(1-\nu)\mu \overleftrightarrow{r}} \begin{bmatrix} \widehat{U}_{11} & \widehat{U}_{12} & \widehat{U}_{13} \\ \widehat{U}_{21} & \widehat{U}_{22} & \widehat{U}_{23} \\ \widehat{U}_{31} & \widehat{U}_{32} & \widehat{U}_{33} \end{bmatrix}, \quad \mu = \frac{E}{2(1+\nu)}, \tag{9}$$

$$\widehat{\overline{\mathbf{P}}}_j^*(\mathbf{x}, v, w) = \frac{1}{8\pi(1-\nu)\overleftrightarrow{r}^2} \begin{bmatrix} \widehat{P}_{11} & \widehat{P}_{12} & \widehat{P}_{13} \\ \widehat{P}_{21} & \widehat{P}_{22} & \widehat{P}_{23} \\ \widehat{P}_{31} & \widehat{P}_{32} & \widehat{P}_{33} \end{bmatrix}. \tag{10}$$

Where $\overleftrightarrow{r_1} = P_j^{(1)}(v, w) - x_1$, $\overleftrightarrow{r_2} = P_j^{(2)}(v, w) - x_2$, $\overleftrightarrow{r_3} = P_j^{(3)}(v, w) - x_3$, $\overleftrightarrow{r} = \sqrt{\overleftrightarrow{r_1}^2 + \overleftrightarrow{r_2}^2 + \overleftrightarrow{r_3}^2}$.

The individual elements in the matrices (9) and (10) in the explicit form can be found in [8]. Integrands (9) and (10) in identity (8) are very similar to functions (2) and (3). The main difference is that functions (9) and (10) require, besides surface patches, coordinates of points $\mathbf{x} = \{x_1, x_2, x_3\}$ in the domain where solutions are searched.

In parallel implementation of PIES only the most time-consuming functions were parallelized: procedure of the system generation - equations (1), (2), (3) and searching solutions in domain - equations (8), (9), (10). It is strictly connected with numerical calculation of integrals. Solving the system of algebraic equations is still calculated in serial way.

Parallel part of computations is as follows: 1. compute table of parameters η from (4) for all collocation points and Gauss-Legendre quadrature coefficients (CUDA kernel), 2. compute tables of integrands (2) and (3) (CUDA kernel), 3. compute table of integrals (1) (CUDA kernel). Table of integrals is sent to CPU where system of algebraic equations is built and solved in serial way. Solutions of the system are sent back to GPU and second part of parallel computations are started: 1. compute tables of integrands (9) and (10) (CUDA kernel), 2. compute solutions (8) (CUDA kernel).

3 General-Purpose GPU - CUDA Technology

The architecture of graphics processing units (GPUs) significantly differs from central processing units (CPUs). GPU is composed of multiple floating point units (FPU) and arithmetic and logic units (ALUs). It is connected with the nature of performed calculations - the same operations are executed in parallel on large amounts of data. Using a lot of pixels, texels or vertices is typical in graphics applications, therefore GPUs are rated as SIMD (single instruction, multiple data).

Comparing to traditional methods of GPGPU programming (eg. OpenCL), NVidia CUDA [1] is much simpler and more intuitive. CUDA is based on C programming language and allows to code GPGPU in parallel computing paradigm. Serial part of CUDA application code is running in CPU (called host) and parallel part in GPU (called device). Data flow in CUDA is shown in Fig. 1. It is divided into four basic steps: 1. initiation of the program (host), 2. copy data from host to device, 3. calculations in the GPU, 4. copy data from device to host.

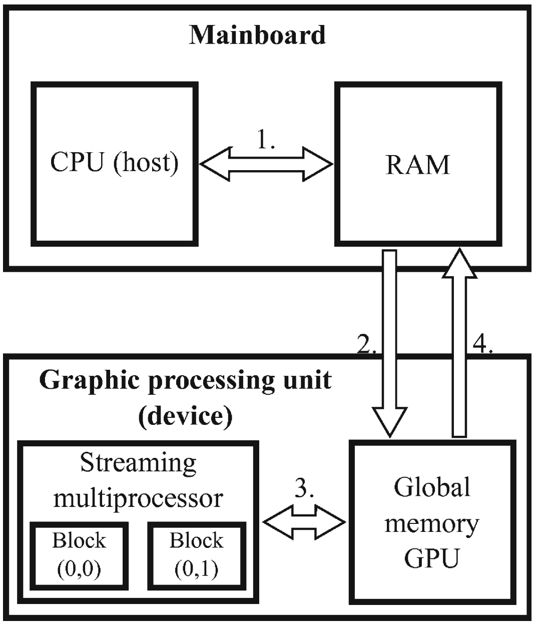


Fig. 1. Data flow diagram in CUDA

CUDA is based on use of scalable multi-threaded array streaming multiprocessors (SMs). Each multiprocessor consists of eight CUDA cores (general-purpose scalar processors), a multithreaded instruction unit and shared memory. A scalar processor can compute one arithmetic operation of add or multiply per clock cycle and two in the case of multiply-and-add operation [1].

The most popular approach to programming CUDA devices is connected with division of the problem into smaller problems, which instructions form functions called kernels. Kernels are executed on a part of threads divided into blocks. Each 32 threads in the block are called warp. All threads within a block can communicate with one another and be synchronized using the synchronization functions. In order to use a larger number of threads within one kernel the blocks of threads are grouped into grids.

Threads can retrieve data from different types of GPU memory. Each thread has access to GPU global memory, which has large capacity, but low bandwidth. Another type is local memory - private to each thread. It has the same parameters as global memory. It stores the variables that should be located in registers. However, due to the small size of the memory of registers (8192 bytes per multiprocessor) when variables occupy too much space, the compiler automatically move some of them to local memory.

Threads within the block can use the software-managed shared memory stored on the chip (called on-chip memory). It is shared by all threads in a single block and has size up to 48 KB. It is much faster than global memory. There is constant memory as well as memory of textures.

CUDA is a very convenient tool to parallelize computations using GPUs, however optimization of application performance is quite difficult. First of all, a programmer should focus on three main problems: memory access as well as code optimization, and configuration of kernel boot parameters.

Another problem, important from the scientific computation point of view, is floating point arithmetic precision. The newest graphics cards allow to use double-precision floating-point operations, older ones - only single-precision. It is particularly important for the high precision calculations.

4 Testing Example

The testing example concerns the problem described by Navier-Lamé equations shown in Fig. 2a. The considered element is firmly fixed at the bottom and subjected to a uniform normal load $p = 1MPa$ acting along the upper part of left side. The values of material constants selected for the calculation are Young's modulus $E = 1MPa$ and Poisson's ratio $\nu = 0.3$.

The shape of the boundary is modelled by 7 rectangular Bézier surfaces of the third degree (curved boundary fragments) and 6 flat rectangular Coons surfaces. A complete declaration of the boundary defined in PIES by 13 surfaces requires 112 points (control points of Bézier and corner points of Coons surfaces). In Fig. 2a, for clarity, only four main points of each surface are marked. On each surface we have defined the same number of collocation points (from 16 to 64) and finally have solved the system of 624 to 2496 algebraic equations.

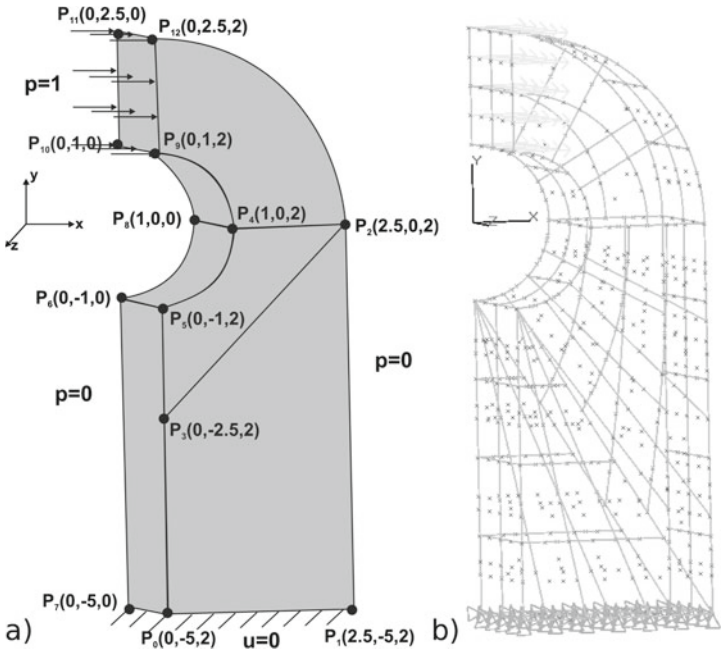


Fig. 2. Considered geometry modelled in a) PIES, b) BEM (BEASY)

NVidia QuadroFX 580 works at 1.125 GHz with 512 MB 128-bit GDDR3 memory (4 streamline multiprocessors each composed by 8 CUDA cores) was used during tests. It is older graphic card which allows to use only the single-precision floating-points operations. Its peak performance is 108 GFlops. Intel Xeon E5507 (4 cores, 4 threads, 2.26 GHz, 4 MB cache memory, peak performance 36.26 GFlops) was used to solve serial version of PIES. Originally, PIES in serial (written in pure C++) implementation running only on CPU uses the double-precision floating-points operations. However, for comparative purposes, the authors present results for single-precision serial version.

Serial version of PIES was compiled using g++ 4.4.3 with standard settings, whilst parallel PIES by nvcc V0.2.1221 CUDA 5.0 release with standard settings, as well. Numerical tests were carried out in 64-bit Ubuntu Linux operation system (kernel 2.6.37).

In order to verify an accuracy of the results obtained by parallel (CUDA) implementation of PIES it was made a comparison with the solutions obtained by PIES in serial implementation and commercial application of BEM BEASY. Therefore, the considered geometry was modelled in BEASY (Fig. 2b), where the mesh for which the minimum number of elements gives stable numerical solutions was applied. Finally, it were used 104 quadrilateral and triangular quadratic elements defined by 475 nodes. The number of algebraic equations that had to be solved was 1425.

Table 1. Comparison of accuracy of the results

No. of quadrature coefficients	No. of collocation points	Relative error norm (in [%])					
		Serial version of PIES (C++)			Parallel version of PIES (CUDA)		
		x	y	z	x	y	z
32	16	0.538	0.880	0.844	0.538	0.880	0.633
	25	0.442	0.406	0.554	0.442	0.407	0.565
	36	0.184	0.122	0.088	0.182	0.121	0.126
	49	0.247	0.172	0.171	0.243	0.168	0.037
	64	0.154	0.118	2.223	0.150	0.116	2.000
64	16	0.101	0.372	0.263	0.138	0.416	0.122
	25	0.114	0.135	0.088	0.087	0.133	0.039
	36	0.241	0.217	0.121	0.191	0.174	0.168

4.1 Comparison of Accuracy of the Results

In order to verify an accuracy of the results, it were computed relative error norm L_2 between solutions obtained by parallel version of PIES and BEASY (treated as exact). This is connected with the lack of analytical solutions of the problem. Solutions from two cross-sections were analyzed, i.e. $x = 1.5, -4.5 \leq y \leq 1.0, z = 1.0$ and $0.25 \leq x \leq 2.25, y = -2.0, z = 1.0$. Error norm L_2 was computed using the following formula [4]:

$$\| e \|_{L_2} = \frac{1}{\left|\overline{U}_i\right|_{max}} \sqrt{\frac{1}{K} \sum_{i=1}^{K} \left(U_i - \overline{U}_i\right)^2} \tag{11}$$

where: K - is the number of obtained numerical solutions U_i in domain, whereas $\overline{U}_i$ - appropriate solutions from BEASY

The values of the obtained error norms are presented in Table 1. Tests were carried out for different numbers of collocation points and different number of Gauss-Legendre quadrature coefficients. The coefficients appear in solving PIES during calculation of matrix elements which require numerical integration. Both serial and parallel version of PIES were tested for single-precision floating-point operations.

As it can be seen from the Table 1, both serial and parallel version of PIES gives results close to MEB and almost the same relative error norms. Despite use of single-precision floating-point numbers in CUDA and serial PIES, there was no significant decrease in accuracy of the obtained solutions. It should be noted that with other examples results may be a bit worse than presented one. It is possible that floating points errors partly compensate a true error of calculations.

4.2 Comparison of Applications Performance

The results of increased speed of parallel version of PIES compared to serial version of PIES is presented in Table 2. Tests were carried out for different num-

Table 2. Speedup of parallel PIES

No. of quadrature coefficients	No. of collocation points	Application execution time (in [s]) Serial version of PIES (C++)	Application execution time (in [s]) Parallel version of PIES (CUDA)	Speedup
32	16	77.50	5.97	12.98
	25	185.72	14.75	12.59
	36	391.23	33.50	11.68
	49	748.07	70.01	10.69
	64	1302.17	136.70	9.53
64	16	303.01	18.40	16.47
	25	722.22	43.67	16.54
	36	1507.90	92.26	16.34

bers of collocation points and different number of Gauss-Legendre quadrature coefficients. Both serial and parallel version of PIES use single-precision floating point operations.

Parallel version of PIES is up to sixteen times faster than serial one. It should be noticed that solving the system of algebraic equations is still computed in serial way. There is no application optimization in terms of use high-speed CUDA device memory.

5 Conclusions

The paper presents the possibility of acceleration of numerical calculations in solving 3D boundary problems modelled by Navier-Lamé equations. It has been achieved by PIES parallelized using CUDA.

Results of solving boundary value problems using parallel version of PIES are obtained up to sixteen times faster than serial one. It should be noticed that any GPU memory optimization was not performed and NVidia QuadroFX 580 is rather old and a bit slow graphics card.

Despite the use of single-precision floating-point numbers in CUDA, parallel version of PIES gives results almost the same as single-precision serial version of PIES. Solutions are very close to ones obtained using commercial application of BEM (BEASY), as well.

This paper is one of the first attempt to use CUDA technology in 3D boundary value problems solved by PIES. Results strongly suggest the chosen direction of studies should be continued.

Acknowledgements. The scientific work is founded by resources for science in the years 2010–2013 as a research project.

References

1. CUDA, C Programming Guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide/
2. Gottlieb, D., Orszag, S.A.: Numerical Analysis of Spectral Methods: Theory and Applications. SIAM, Philadelphia (1977)
3. Kiss, I., Gyimóthy, S., Badics, Z., Pávó, J.: Parallel realization of the element-by-element FEM technique by CUDA. IEEE Trans. Magn. **48**, 507–510 (2012)
4. Mukherjee, S., Mukherjee, X.Y.: Boundary Methods Elements, Contours and Nodes. CRC Press, Boca Raton (2005)
5. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Kruger, J., Lefohn A.E., Purcell, T.J.: A survey of general-purpose computation on graphics hardware. In: Eurographics 2005 State of the Art Reports, pp. 21–51. Dublin, Ireland (2005)
6. Takahashi, T., Hamada, T.: GPU-accelerated boundary element method for Helmholtz equation in three dimensions. Int. J. Numer. Method Eng. **80**, 1295–1321 (2009)
7. Zieniuk, E.: Bézier curves in the modification of boundary integral equations (BIE) for potential boundary-values problems. Int. J. Solids Struct. **40**, 2301–2320 (2003)
8. Zieniuk, E.: Computational method PIES for Solving Boundary Value Problems (in polish). PWN, Warszawa (2013)
9. Zieniuk, E., Boltuc, A.: Bézier curves in the modeling of boundary geometries for 2D boundary problems defined by Helmholtz equation. J. Comput. Acoust. **14**, 1–15 (2006)
10. Zieniuk, E., Boltuc, A.: Non-element method of solving 2D boundary problems defined on polygonal domains modeled by Navier equation. Int. J. Solids Struct. **43**, 7939–7958 (2006)
11. Zieniuk, E., Boltuc, A., Kuzelewski, A.: Algorithms of identification of multi-connected boundary geometry and material parameters in problems described by Navier-Lamé equation using the PIES. In: Pejas, J., Saeed, K. (eds.) Advances in Information Processing and Protection, pp. 409–418. Springer, New York (2007)
12. Zieniuk, E., Szerszen, K.: Triangular Bézier patches in modelling smooth boundary surface in exterior Helmholtz problems solved by PIES. Arch. Acoust. **34**, 1–11 (2009)
13. Zieniuk, E., Szerszen, K., Boltuc, A.: Convergence analysis of the boundary geometry identification obtained by genetic algorithms in the PIES. In: Saeed, K., Pejas, J., Mosdorf, R. (eds.) BioMetrics, Computer Security Systems and Artificial Intelligence Applications, vol. III, pp. 333–340. Springer, New York (2006)
14. Zieniuk, E., Szerszen, K., Kapturczak, M.: A numerical approach to the determination of 3D stokes flow in polygonal domains using PIES. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011, Part I. LNCS, vol. 7203, pp. 112–121. Springer, Heidelberg (2012)

Modeling and Simulations of Beam Stabilization in Edge-Emitting Broad Area Semiconductor Devices

Mindaugas Radziunas[1](✉) and Raimondas Čiegis[2]

[1] Weierstrass Institute, Mohrenstr. 39, 10117 Berlin, Germany
Mindaugas.Radziunas@wias-berlin.de
[2] Vilnius Gediminas Technical University, Saulėtekio al. 11, 10223 Vilnius, Lithuania
rc@vgtu.lt

Abstract. A 2+1 dimensional PDE traveling wave model describing spatial-lateral dynamics of edge-emitting broad area semiconductor devices is considered. A numerical scheme based on a split-step Fourier method is presented and implemented on a parallel compute cluster. Simulations of the model equations are used for optimizing of existing devices with respect to the emitted beam quality, as well as for creating and testing of novel device design concepts.

Keywords: Broad area device · Traveling wave model · Numerical scheme · Simulations · Beam improvement

1 Introduction

High power high brightness edge-emitting (EE) broad area semiconductor (BAS) lasers and optical amplifiers are compact devices playing a key role in different laser technologies. They have a relatively simple geometry (Fig. 1(a)) allowing an efficient pumping through a broad electric contact on the top of the device and are able to operate at the high power (tens of Watts) regimes. However, once operated at high power regimes, BAS devices suffer from a relatively low quality of the emitted beam which has undesirable broad optical and lateral spectra. A high quality of the beam amplified in BAS amplifiers or generated by BAS lasers is a very important issue of the modern semiconductor laser technology, and there are several BAS device concepts for improving of the emitted beam.

Mathematical modeling, simulations and analysis play a significant role in optimization of existing devices or creation of novel design concepts [1]. Typically, the length (z dimension) and width (x-dimension) of EE BAS devices (see Fig. 1(a)) are in a few millimeter and hundreds of micrometer range, respectively, whereas the height (y dimension) of the active zone where the optical beam is generated and amplified is, typically, not larger than a micrometer. Since full 3-dimensional dynamical simulations of semiconductor devices with different spatial and temporal scales is not possible, we replace all y-dependent

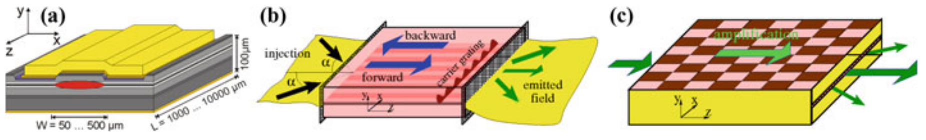


Fig. 1. Schemes of different EE BAS device configurations. (a): standard EE BAS laser. (b): BAS laser with a dual angular plane wave injection. (c): BAS amplifier with a spatially periodic electrical contact.

quantities by some effective vertical averages. To simulate the generation and/or propagation of the optical fields along the cavity of EE BAS devices we use a 2+1 dimensional system of PDEs, described briefly in this paper. The model is based on the traveling wave (TW) equations for counter-propagating and laterally diffracted slowly varying optical fields which are coupled to the ODE for induced polarizations and diffusive rate equation for carrier densities [2,3]. The well-posedness of this model was studied in [4], while different algorithms used for the numerical integration of the model were considered in [5–7].

Precise dynamic simulations of long and broad or tapered devices and tuning/optimization of the model with respect to one or several parameters, require huge CPU time and memory resources. A proper resolution of rapidly oscillating fields in typical BAS devices on a sufficiently large optical frequency range requires a fine space (10^6–10^7 mesh points) and time (up to 10^6 points for typical 5 ns transients) discretization. Dynamic simulations of such devices can easily take one or even several days of computations on a single processor computer. Some speedup of computations can be achieved by using problem-dependent relations of the grid steps, including also variable steps in the lateral dimension. All these grid optimizations, however, are not sufficient when one- or a few-parameter studies with the simulation times up to 1000 ns should be performed. It is obvious, that the required computations in an acceptable time can only be done by means of parallel computers and parallel solvers.

In this paper we present a split-step Fourier method based numerical algorithm for the integration of the 2+1 dimensional traveling wave model of BAS devices. It was implemented on the parallel compute cluster at the Weierstrass Institute in Berlin and was successfully used for simulations of different BAS devices with an improved quality of the beam [3,8–11]. In this paper we present two examples of such BAS devices (see Fig. 1(b) and (c)) which were proposed in our theoretical papers [12–14]. In the first device, a pair of coherently injected plane waves at the adjoint angles to the laser axis (Fig. 1(b)) can create a periodic carrier grating, which in turn can suppress all but one lateral modes of the laser [12,13]. The second device has periodically modulated (PM) electrical contact or active zone in both spatial directions (Fig. 1(c)), what can lead to a significant improvement of the amplified beam in BAS amplifiers [14].

2 Mathematical Model

After an appropriate scaling, the traveling wave (TW) model for longitudinal-lateral dynamics of the complex slowly varying amplitudes of the counter-propagating fields $E^{\pm}(z,x,t)$, polarization functions $P^{\pm}(z,x,t)$ and real carrier density function $N(z,x,t)$ can be written as follows [7]:

$$\begin{aligned}
&\left(\frac{\partial}{\partial t} \pm \frac{\partial}{\partial z}\right) E^{\pm} = -\frac{i}{2}\frac{\partial^2}{\partial x^2}E^{\pm} - i\left[\beta(N,\|E\|^2) - i\frac{\mathcal{D}}{2}\right] E^{\pm} - i\kappa^{\mp}E^{\mp},\\
&\frac{\partial}{\partial t}P^{\pm} = i\overline{\omega}P^{\pm} + \overline{\gamma}\left(E^{\pm} - P^{\pm}\right), \qquad (1)\\
&\frac{1}{\mu}\frac{\partial}{\partial t}N = D\frac{\partial^2}{\partial x^2}N + I(z,x) - R(N) - \Re e\sum_{\nu=\pm} E^{\nu *}\left[G(N,|E^{\pm}|^2) - \mathcal{D}\right]E^{\nu},
\end{aligned}$$

where $\|E\|^2 = |E^+|^2 + |E^-|^2$ is proportional to the local field intensity, whereas the operators β, $\mathcal{D}$ and functions G, $\tilde{n}$, R denote the propagation factor, the Lorentzian approximation of the material gain dispersion, the gain peak value, the refractive index change, and the spontaneous recombination, respectively:

$$\begin{aligned}
&\beta(N,|E^{\pm}|^2) = \Delta - \tilde{n}(N) + \frac{i\left(G(N,\|E\|^2) - \alpha\right)}{2}, \quad \mathcal{D}E^{\pm}{=}\overline{g}\left(E^{\pm} - P^{\pm}\right),\\
&G(N,\|E\|^2) = \frac{g' N_{tr}}{1+\varepsilon\|E\|^2}\log\left(\frac{\max(N,N_*)}{N_{tr}}\right), \quad \tilde{n}(N) = 2\sigma N_{tr}\sqrt{N/N_{tr}}, \qquad (2)\\
&R(N) = AN + BN^2 + CN^3.
\end{aligned}$$

In general, this model should be considered in the unbounded region $Q = Q_{z,x} \times (0,T]$, where $Q_{z,x} = \{(z,x) :\ (z,x) \in (0,L) \times \mathbf{R}\}$ is the spatial domain, L represents the device length, x is the coordinate of the unbounded lateral axis of the device, and T is the length of the time interval where we perform the integration. In our numerical simulations we choose a large enough lateral interval $[-X,X]$ containing the considered BAS device and assume that the field and carrier density functions $E^{\pm}$ and N are periodic along the lateral axis:

$$E^{\pm}(z,x+2X,t) = E^{\pm}(z,x,t), \quad N(z,x+2X,t) = N(z,x,t), \quad (z,x,t) \in Q. \quad (3)$$

This assumption restricts our considerations of the model equations to the truncated domain $Q^X = Q^X_{z,x} \times (0,T]$, $Q^X_{z,x} = \{(z,x) :\ (z,x) \in (0,L) \times [-X,X]\}$. The boundary conditions for the optical fields $E^{\pm}$ at the device facets $(z,x) \in 0 \times [-X,X]$ and $(z,x) \in L \times [-X,X]$ in (1) are given by

$$\begin{aligned}
&E^-(L,x,t) = r_1(x)\,E^+(L,x,t) + F\left[E^+(L,\cdot,t-\tau)\right],\\
&E^+(0,x,t) = r_0(x)\,E^-(0,x,t) + a(x,t), \qquad (x,t) \in [-X,X] \times [0,T],
\end{aligned} \qquad (4)$$

where $r_{0,1}$, a and F are the field amplitude reflectivity coefficients, the complex amplitude of the optical field injection, and another optical source determined by the reinjected delayed optical field [10], respectively. The initial conditions

$$E^{\pm}(z,x,0) = E^{\pm}_0(z,x), \quad P^{\pm}(z,x,0) = P^{\pm}_0(z,x), \quad N(z,x,0) = N_0(z,x) \qquad (5)$$

are defined for $(z,x) \in Q^X_{z,x}$. If properly stated, they are not important, since after some transients the trajectories approach one of the stable attractors.

The coefficients $\kappa^{\pm}$, $\varDelta$, α, g', σ, N_{tr}, N_*, ε, μ, D, I, A, B and C represent the complex field coupling due to the Bragg grating, the static detuning due to the built-in refractive index profile, the internal losses of the field, the differential gain, the differential index, the carrier density at the transparency, the gain clamping carrier density, the nonlinear gain compression, the scaling factor related to the ratio of the photon and carrier life times, the carrier diffusion coefficient, the current injection density, and three recombination factors, respectively. Finally, $\overline{g}$, $\overline{\omega}$ and $\overline{\gamma}$ denote the amplitude, the central frequency and the half width at half maximum of the Lorentzian fitting of the gain profile.

Most of the parameters are spatially non-homogeneous and even discontinuous depending on the device geometry. More details about the meaning and typical values of all parameters can be found in [2,3]. Normalization of the equations and typical values of the normalized parameters are given in [7]. It is noteworthy, that $\overline{\gamma} \approx 10^2 \div 10^3$ and $\mu \approx 10^{-3}$ represent the fast relaxation of the polarizations $P^{\pm}$ and slow dynamics of the carrier density N, respectively. Typical size of the dimensionless domain is determined by $X \approx 5 \div 30$ and $L \approx 1 \div 10$, whereas $D \approx 0.5$ and most of other parameters are of order $\mathcal{O}(1)$.

3 Numerical Scheme

The computation domain Q^X is discretized using a uniform in space and time grid $Q^X_h = Q^X_{h,z,x} \times \omega_{h,t}$, where $Q^X_{h,z,x} = \omega_{h,z} \times \omega_{h,x}$, and

$$\begin{aligned}
\omega_{h,x} &= \{x_j : \quad x_j = j\, h_x, \quad j = -J/2, \dots, J/2-1, \; h_x = 2X/J\},\\
\omega_{h,z} &= \{z_k : \quad z_k = k\, h, \quad k = 0, \dots, K, \; h = L/K\},\\
\omega_{h,t} &= \{t_m : \quad t_m = m\, h, \quad m = 0, \dots, M, \; M = T/h\}.
\end{aligned}$$

The time discretization step h is equal to the spatial step in z-direction, what allows an accurate optical field propagation along the characteristic lines $z \pm t =$ const. We note, that h is the maximal allowed time step: its further increasing violates the Courant-Friedrichs-Lewy (CFL) condition and, therefore, stability of the numerical schemes.

All spatially depending parameters $P(z,x)$, spatially and temporarily depending functions $F(x,t)$ and unknown functions $U(z,x,t)$ in Eqs. (1)–(5) are approximated by their grid analogs defined on Q^X_h:

$$P_{k,j} = P(z_k, x_j), \qquad F^m_j = F(x_j, t_m), \qquad U^m_{k,j} \approx U(z_k, x_j, t_m).$$

When constructing numerical schemes we exploit a discrete Fourier transform of complex and real laterally-periodic functions $U(z,x,t)$, where $U = E^{\pm}$ or $U = N$. Namely, we assume that a complex function $U_j(z,t) := U(z,x_j,t)$ (representing the fields E^+ and E^-) on the uniform lateral mesh $\omega_{h,x}$ can be expressed as a linear combination of the orthonormal grid-functions $e^{i\pi\ell x_j/X}|_{\ell=-J/2}^{J/2-1}$:

$$U_j(z,t) = \left[\mathcal{F}^{-1}\left(\widehat{U}_\ell(z,t)\big|_{\ell=-J/2}^{J/2-1}\right)\right]_j := \frac{1}{J}\sum_{\ell=-J/2}^{J/2-1}\widehat{U}_\ell(z,t)e^{i\pi\ell x_j/X}, \qquad (6)$$

where the Fourier coefficients $\widehat{U}_\ell(z,t)$ are defined as

$$\widehat{U}(z,t) = \left[\mathcal{F}\left(U_j(z,t)\big|_{j=-J/2}^{J/2-1}\right)\right]_\ell := \sum_{j=-J/2}^{J/2-1} U_j(z,t)e^{-i\pi\ell x_j/X}. \qquad (7)$$

These transforms are used for the approximation of $\frac{\partial^2}{\partial x^2}U$ at any grid point x_j by the second derivative of the trigonometric interpolant:

$$\frac{\partial^2}{\partial x^2}U(z,x_j,t) \approx \frac{1}{J}\sum_{\ell=-J/2}^{J/2-1}\left(-\frac{\pi^2\ell^2}{X^2}\right)\widehat{U}_\ell(z,t)e^{i\pi\ell x_j/X}. \qquad (8)$$

When U represents the carrier density N and is *real*, it can be expressed as a linear combination of the real orthogonal grid-functions $\cos(\pi\ell x_j/X)|_{\ell=0}^{J/2}$, and $\sin(\pi\ell x_j/X)|_{\ell=1}^{J/2-1}$. An equivalent *complex* expression of such combination can be written as (6) with the *complex* Fourier coefficients (7) satisfying the relations $\widehat{N}_{-J/2} = \widehat{N}^*_{-J/2}$ and $\widehat{N}_\ell = \widehat{N}^*_{-\ell}$, $\ell = 0,\dots,J/2-1$.

3.1 Splitting Scheme

The TW model (1)–(5) is integrated numerically using a splitting scheme, where the lateral field diffraction and carrier diffusion are resolved with the fast Fourier transform, and the remaining coupled hyperbolic system in (1) is integrated along the characteristics using finite differences. The stiff ODE for the polarization functions $P^\pm$ in (1) ($\overline{\gamma}$ is large!) is resolved using an exponentially weighted scheme with the values for $E^\pm$ at the same time layer, which ensures, that $\lim_{\overline{\gamma}\to\infty} P^\pm = E^\pm$.

Let us assume, that the grid functions $E^{\pm,m}_{k,j}$, $P^{\pm,m}_{k,j}$ and $N^m_{k,j}$ are known for the time layer t^m. In the time-stepping algorithm we split the diffraction, diffusion processes and the nonlinear interaction. To find the grid functions at the new time layer t^{m+1} we proceed as follows. In the first step of our algorithm we consider only the nonlinear interaction and make a simple prediction of the carrier density at the new time layer:

$$\frac{\widetilde{N}^{m+1}_{k,j} - N^m_{k,j}}{\mu h} = -\left(G(N^m_{k,j}, \|E^m_{k,j}\|^2) - \overline{g}\right)\|E^m_{k,j}\|^2 - \overline{g}\,\Re e\sum_{\nu=\pm} E^{\nu,m*}_{k,j}P^{\nu,m}_{k,j} \qquad (9)$$
$$+ I_{k,j} - \frac{\widetilde{N}^{m+1}_{k,j}R(N^m_{k,j})}{N^m_{k,j}}, \quad k = 0,\dots,K, \quad j = -J/2,\dots,J/2-1.$$

We use a simple implicit-explicit linearized scheme at this step.

In the next step we neglect the field diffraction and find intermediate approximations for the optical fields and new polarization functions:

$$\begin{aligned}
&\frac{\widetilde{E}^{\pm,m+1}_{k,j} - E^{\pm,m}_{k\mp1,j}}{h}\\
&\quad= -i\frac{\beta\left(\widetilde{N}^{m+1}_{k,j}, \|E^m_{k,j}\|^2\right)\widetilde{E}^{\pm,m+1}_{k,j} + \beta\left(N^m_{k\mp1,j}, \|E^m_{k\mp1,j}\|^2\right)E^{\pm,m}_{k\mp1,j}}{2}\\
&\qquad - \frac{\overline{g}_{k,j}(\widetilde{E}^{\pm,m+1}_{k,j} - P^{\pm,m+1}_{k,j}) + \overline{g}_{k\mp1,j}(E^{\pm,m}_{k\mp1,j} - P^{\pm,m}_{k\mp1,j})}{4}\\
&\qquad - i\frac{\kappa^{\mp}_{k,j}\widetilde{E}^{\mp,n+1}_{k,j} + \kappa^{\mp}_{k\mp1,j}E^{\mp,m}_{k\mp1,j}}{2}, \qquad k, k\mp1 \in \{0,\dots,K\};\\
&\widetilde{E}^{+,m+1}_{0,j} = r_{0,j}\widetilde{E}^{-,m+1}_{0,j} + a^{m+1}_j, \widetilde{E}^{-,m+1}_{K,j} = r_{1,j}\widetilde{E}^{+,m+1}_{K,j} + F_h\left[E^{+,m+1-\frac{\tau}{h}}_{K,\cdot}\right], \quad (10)\\
&P^{\pm,m+1}_{k,j} = \frac{\overline{\gamma}_{k,j}\left(1 - e^{(i\overline{\omega}_{k,j} - \overline{\gamma}_{k,j})h}\right)}{\overline{\gamma}_{k,j} - i\overline{\omega}_{k,j}}\widetilde{E}^{\pm,m+1}_{k,j} + e^{(i\overline{\omega}_{k,j} - \overline{\gamma}_{k,j})h}P^{\pm,m}_{k,j}, \; k = 0,\dots,K,\\
&\qquad j = -j/2,\dots,j/2-1.
\end{aligned}$$

We note, that the scheme above is linear with respect to $\widetilde{E}^{\pm,m+1}_{k,j}$ and $P^{\pm,m+1}_{k,j}$ and can be separately resolved for each $k = 0,\dots,K$.

In the final step of our algorithm we take into account the carrier diffusion and field diffraction. Namely, we solve the linear equations

$$\frac{\partial}{\partial t}N = \mu D\frac{\partial^2}{\partial x^2}N, \qquad \left(\frac{\partial}{\partial t} \pm \frac{\partial}{\partial z}\right)E^{\pm} = -\frac{i}{2}\frac{\partial^2}{\partial x^2}E^{\pm}$$

within the time (and space) interval of length h, whereas the initial conditions are given by the previously obtained estimates $\widetilde{N}$ and $\widetilde{E}^{\pm}$. To integrate these equations we use lateral discretizations of the functions N and $E^{\pm}$, approximate their second lateral derivatives by (8) and solve the resulting systems of the differential equations in the (lateral) Fourier domain:

$$\widehat{N}_\ell(z, t+h) = e^{-\mu D\frac{\pi^2\ell^2}{X^2}h}\widehat{N}_\ell(z,t), \qquad \widehat{E}^{\pm}_\ell(z \pm h, t+h) = e^{i\frac{\pi^2\ell^2}{2X^2}h}\widehat{E}^{\pm}_\ell(z,t).$$

The inverse discrete Fourier transform (6) and the discretization of the functions N and $E^{\pm}$ along the longitudinal z direction give us the following equations, which completes the description of our numerical scheme:

$$\begin{aligned}
N^{m+1}_{k,j} &= \frac{1}{J}\sum_{\ell=-J/2}^{J/2-1}\left[e^{-\mu D\frac{\pi^2\ell^2}{X^2}h}\sum_{s=-J/2}^{J/2-1}\widetilde{N}^{m+1}_{k,s}e^{-i\frac{2\pi\ell s}{J}}\right]e^{i\frac{2\pi\ell j}{J}},\\
E^{\pm,m+1}_{k,j} &= \frac{1}{J}\sum_{\ell=-J/2}^{J/2-1}\left[e^{i\frac{\pi^2\ell^2}{2X^2}h}\sum_{s=-J/2}^{J/2-1}\widetilde{E}^{\pm,m+1}_{k,s}e^{-i\frac{2\pi\ell s}{J}}\right]e^{i\frac{2\pi\ell j}{J}}, \qquad (11)\\
&\qquad k = 0,\dots,K, \qquad J = -j/2,\dots,j/2-1.
\end{aligned}$$

3.2 Parallelization

Numerical scheme (9)–(11) is well suited for the execution on parallel clusters of computers. We use the domain decomposition technique. To distribute the computational work among different processes $\pi_l|_{l=1}^n$ we decompose the computational grid Q_h^X along the longitudinal z- direction into n non-overlapping sub-grids $Q_h^{X,l}$.

Any process π_l operates on the sub-grid

$$Q_h^{X,l} = \omega_{h,z}^l \times \omega_{h,x} \times \omega_{h,t}, \qquad \omega_{h,z}^l = \omega_{h,z} \cap [(l-1)sh, \min\{(ls-1)h, L\}],$$
$$s = \text{ceil}\,((K+1)/n), \qquad l = 1, \ldots, n.$$

A schematic representation of the full computational grid and its splitting to smaller sub-grids is given in Fig. 2.

Before computing the grid functions at the next time layer (empty bullets in Fig. 2) each process π_l should exchange the current time layer values of $E^\pm$, $P^\pm$ and N at the boundaries of the sub-grid $Q_h^{X,l}$ (light bullets in the same figure) with the adjacent processes π_{l-1} and π_{l+1}. This information is recorded to the specially created ghost grid points (small dots) at the adjacent side of the sub-grids $Q_h^{X,l-1}$ and $Q_h^{X,l+1}$. The left and right ghost points of the sub-grid $Q_h^{X,l}$ in the consequent computations of the process π_l are treated like usual grid points $(z_{(l-1)s-1}, x_j, t^m)$ and (z_{ls}, x_j, t^m) which are not directly accessible by π_l.

The processes π_1 and π_n operating on the end sub-grids $Q_h^{X,1}$ and $Q_h^{X,n}$ have no left or right adjacent sub-grid. The required sub-grid boundary information in these cases is given by the longitudinal boundary conditions (4) including optional optical injection and optical feedback functions $a(x,t)$ and $F[E^+(L,x,t-\tau)]$ (thick empty in-pointing arrows in the same figure). The scalability analysis of the proposed parallel algorithm can be done as in [5]. It proves that the algorithm scales linearly with respect to the number of processors used to solve the given problem.

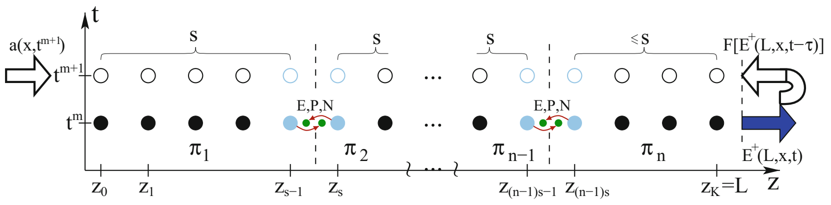


Fig. 2. Scheme of the computational grid (z and t coordinates only). Vertical dashed lines: splitting of the grid Q_h^X to smaller sub-grids $Q_h^{X,l}$. Full and empty bullets: the actual (already computed) and the next time layers, respectively. Arrows: data streams which should be read or recorded by different processes π_l before the next time iteration. Small dots: ghost points of the sub-grid containing an information received from the corresponding border point (light bullets) of the adjacent sub-grid.

In addition to the solution of the scheme (9)–(11) on the sub-grid $Q_h^{X,n}$, the last process π_n records the emitted field $E^+(L, x, t)$ and calculates distributions of the optical feedback (if considered). Thus, the fact that $Q_h^{X,n}$ has, possibly, less grid points then the other sub-grids (the size of $\omega_{h,z}^n$ can be smaller than s) could be advantageous seeking to speed up the simulations.

The numerical scheme (9)–(11) and the parallel algorithm were implemented and executed on a 48 node HP Blade server using the HPMPI library. The nodes are interconnected via Infiniband 4xDDR (20 Gbit/s).

4 Simulations of BAS Devices

Mathematical modeling and fast numerical simulations are a powerful method used in optimization of the existing BAS devices or in creation of the novel design concepts for different real world applications. Below in this section we simulate two theoretically proposed BAS devices (shown also in Fig. 1(b) and (c)) showing an improved quality of the emitted beam.

4.1 Stabilization of a BAS Laser by a Dual Off-Axis Optical Injection

In our theoretical papers [12,13] a new control method of BAS lasers was proposed, which, as we believe, should suppress all but one optical mode, i.e., should stabilize the emitted beam. This control is achieved by a pair of coherent optical plane waves injected into the BAS laser at the adjacent angles to the laser axis (Fig. 1(b)). In the non-scaled model, this optical injection is described by the function

$$a(x, t) = a_0 e^{i(\omega t + \alpha k_0 - \pi/2)} + a_0 e^{i(\omega t - \alpha k_0 + \pi/2)} = 2a_0 e^{i\omega t} \sin(\alpha k_0 x)$$

entering boundary conditions (4). The parameters $\pm\alpha$, k_0, and ω in the expression above denote the free space angles of the injected beams (see Fig. 1(d)), the central wave-vector of the emitted field, and the frequency detuning of the optical injection from the central frequency $k_0 c$ (c: speed of light in vacuum). The factor $|a_0|^2$ is proportional to the intensity of the optical field injected into the laser.

We have performed a series of simulations for the fixed detuning ($\omega = 0$ in this example) and increased intensity of the optical injection (i.e., parameter $|a_0|^2$). The observed laser dynamics (optical spectra, far-fields, field intensities) for different injection intensities is summarized in Fig. 3. Here one can distinguish three qualitatively different regimes, separated by thin vertical lines in Fig. 3. Once the injection intensity is too small, the spatial-temporal dynamics of the system is similar to that one of the free-running BAS laser. This can be recognized by multiple peaks of the optical spectrum (panel (a)), by scattered far-field instants (panel (b)), as well as by a non-stationary output field (differing minimal and maximal intensities in panel (c)). For moderate and large injected

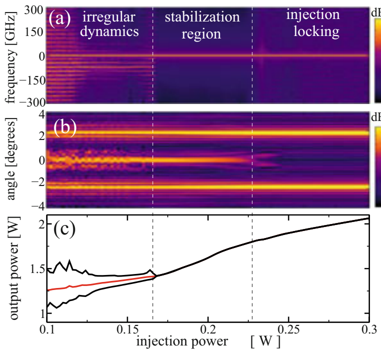


Fig. 3. Stabilization of the BAS laser by the optical injection. (a): mapping of the optical spectra, (b): mapping of the far-fields computed at some time instant, and (c): maximal, minimal and mean power of the emitted field for the increased injection power and fixed $\omega = 0$.

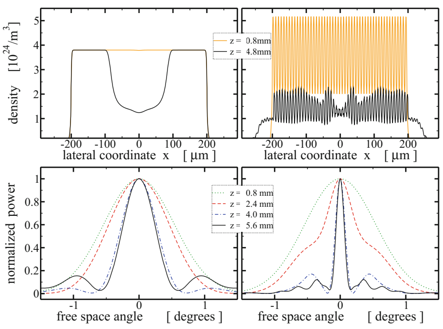


Fig. 4. Amplification of the optical beam in the equally biased conventional (left) and the PM EE BAS amplifiers with $(d_x, d_z) = (8, 400)\,\mu$m, and $\mathcal{Q} = 1.02$ (right). First and second rows show lateral distributions of the carrier density and central part of the far-fields computed for selected longitudinal positions z.

field intensities the laser operates at a continuous wave regime (a single spectral line in panel (a) and coinciding minimal and maximal powers in panel (c)). An inspection of the far-fields at these injections, however, allows us to distinguish two different regimes. Namely, for moderate injections we have a stationary state which has a well pronounced central angular component (a stabilized mode of the laser), whereas for larger injections only the angular components corresponding to the injected beam angles α are present. In this regime our BAS laser is operating like an amplifier for the injected beams, but does not generate light by itself.

4.2 BAS Amplifiers with Periodically Modulated Electrical Contacts

An elegant way to improve the lateral beam profile in EE BAS amplifiers was suggested in the recent theoretical work [14]. It was shown, that a periodic modulation of the gain and refractive index in both longitudinal and lateral directions (see Fig. 1(c)) can lead to a significant compression of the far-fields, what is desirable in the real world applications.

A crucial condition for the desired beam shaping is a proper choice of the lateral and longitudinal modulation periods d_x and d_z, which in the non-scaled model should satisfy the relation $\mathcal{Q} = \frac{d_x^2 k_0 n_b}{\pi d_z} \approx 1$, where n_b is the background refractive index in the semiconductor device (typically about $3 \div 3.5$). The mathematical model used in [14], however, was oversimplified: it was neglecting a strong nonlinear interaction of carriers and optical fields in high-power devices, i.e., was only suitable for simulations of very small fields (and polarizations) which have no impact to the carrier distribution (see the carrier rate equation in Eq. (1)).

In the present work we have performed simulations of a standard BAS amplifier (left panels of Fig. 4) and of a BAS amplifier with the PM electrical contact (right panels of the same figure) operating in moderate and high power regimes. In these regimes the carrier distribution is strongly depleted, causing lateral irregularities in the carrier (i.e., gain and refractive index) modulation amplitudes: see the black curves in the first row panels of Fig. 4, representing the carrier densities at $z = 4.8\,\mathrm{mm}$. The simulations have shown, that the desired beam shaping in the PM BAS amplifiers can be also obtained using our more realistic modeling approach: compare the far-fields of the simple (left) and PM amplifiers (right) at the lower row panels of Fig. 4. It is noteworthy, that even though a part of the field amplified in the PM device is radiated at $\approx\pm 7.2°$ side band components, the intensity of the remaining central angle field still can be higher than that one of the field amplified in the conventional BAS device.

In conclusion, we have presented a numerical scheme for the 2+1 dimensional PDE model describing the dynamics of BAS devices, and discussed its implementation on the parallel cluster of computers. The mathematical model and the numerical scheme were applied for the study of the beam stabilization in different configurations of BAS devices.

Acknowledgments. The work of M. Radziunas was supported by DFG Research Center Matheon. The authors would like to thank Mark Lichtner for implementation of the numerical schemes on the parallel compute cluster.

References

1. Wenzel, H.: Basic aspects of high-power semiconductor laser simulation. IEEE J. Sel. Top. Quantum Electron. **19**(5), 1502913 (2013)
2. Bandelow, U., et al.: Impact of gain dispersion on the spatio-temporal dynamics of multisection lasers. IEEE J. Quantum Electron. **37**, 183–188 (2001)
3. Spreemann, M., et al.: Measurement and simulation of distributed-feedback tapered master-oscillators power-amplifiers. IEEE J. Quantum Electron. **45**, 609–616 (2009)
4. Lichtner, M., Radziunas, M., Recke, L.: Well posedness, smooth dependence and center manifold reduction for a semilinear hyperbolic system from laser dynamics. Math. Meth. Appl. Sci. **30**, 931–960 (2007)
5. Čiegis, R., Radziunas, M., Lichtner, M.: Numerical algorithms for simulation of multisection lasers by using traveling wave model. Math. Model. Anal. **13**, 327–348 (2008)
6. Laukaitytė, I., et al.: Parallel numerical algorithm for the traveling wave model. In: Čiegis, R., Henty, D., Kagstrom, B., Žilinskas, J. (eds.) Parallel Scientific Computing and Optimization, vol. 27, pp. 237–251. Springer, New York (2009)

7. Čiegis, R., Radziunas, M.: Effective numerical integration of traveling wave model for edge-emitting broad-area semiconductor lasers and amplifiers. Math. Model. Anal. **15**, 409–430 (2010)
8. Radziunas, M., et al.: Mode transitions in distributed-feedback tapered master-oscillator power-amplifier. Opt. Quantum Electron. **40**, 1103–1109 (2008)
9. Tronciu, V.Z., et al.: Improving the stability of distributed-feedback tapered master-oscillator power-amplifiers. Opt. Quantum Electron. **41**, 531–537 (2009)
10. Jechow, A., et al.: Stripe-array diode-laser in an off-axis external cavity: theory and experiment. Opt. Express **17**, 19599–19604 (2009)
11. Tronciu, V.Z., et al.: Amplifications of picosecond laser pulses in tapered semiconductor amplifiers: numerical simulations versus experiments. Opt. Commun. **285**, 2897–2904 (2012)
12. Radziunas, M., Staliunas, K.: Spatial rocking in broad area semiconductor lasers. Europhys. Lett. **95**, 14002 (2011)
13. Radziunas, M., Staliunas, K.: Spatial "rocking" for improving the spatial quality of the beam of broad area semiconductor lasers. In: SPIE Proceeding, vol. 8432, p. 84320Q (2012)
14. Herrero, R., et al.: Beam shaping in spatially modulated broad area semiconductor amplifiers. Opt. Lett. **37**, 5253–5255 (2012)

Concurrent Nomadic and Bundle Search: A Class of Parallel Algorithms for Local Optimization

Costas Voglis[1,2](✉), Dimitrios G. Papageorgiou[3], and Isaac E. Lagaris[2]

[1] Nodalpoint Systems LTD, Athens, Greece
voglis@cs.uoi.gr
[2] Department of Computer Science, University of Ioannina, 1186, 45110 Ioannina, Greece
[3] Department of Materials Science and Engineering, University of Ioannina, 1186, 45110 Ioannina, Greece

Abstract. We present a family of algorithms for local optimization that exploit the parallel architectures of contemporary computing systems to accomplish significant performance enhancements. This capability is important for demanding real time applications, as well as, for problems with time–consuming objective functions. The proposed concurrent schemes namely *nomadic* and *bundle* search are based upon well established techniques such as quasi-Newton updates and line searches. The parallelization strategy consists of (a) distributed computation of an approximation to the Hessian matrix and (b) parallel deployment of line searches on different directions (bundles) and from different starting points (nomads). Preliminary results showed that the new parallel algorithms can solve problems in less iterations than their serial rivals.

Keywords: Parallel local search · Nomadic search · Concurrent search · Nested parallelism · Line search · Quasi-Newton

1 Introduction

Optimization has broad practical applications in engineering, science and management. Many of these may either have expensive function evaluations or require real–time response. For example we refer to aircraft design, molecular modeling, space trajectory planning, optimal sea routing etc. High performance parallel computing can provide powerful tools for solving optimization problems.

Nowadays the computing power of a single core CPU has reached a premium that is improving at a very slow pace. The direction for performance enhancement has turned from pushing the CPU clock higher up, towards creating multi-core parallel architecture systems. However these hardware developments alone cannot change the picture overnight. Suitable software must be developed based on algorithms that can exploit the parallel features of the hardware in order to reach the desired performance levels. In critical real time applications, an untimely (delayed) result is at best useless if not costly or catastrophic. In addition, large scale tough problems with expensive objectives, left aside or abandoned as hopeless goals due to the extremely long computational times they required, are now being reconsidered in the light of parallel processing.

Parallelizing a sequential algorithm usually results in minor performance gains and often the new algorithm is not equivalent to the original. However, an algorithm designed afresh aiming to exploit parallelism, is naturally expected to attain high levels of performance. Monte–Carlo methods are inherently parallel and their implementation on parallel systems is quite straightforward. Global optimization methods are also parallelizable rather easily due to the nature of the problem itself. The case of local optimization is quite hard and indeed very few inherently parallel methods exist. Most of them are advertised as parallel by adopting a parallel linear system solver. Algorithms of that kind are not genuinely parallel optimization algorithms, since they treat in parallel only the bookkeeping operations and not the calls to the objective function. The work of Chen and Han [1], describes a method called "Parallel Quasi-Newton", which handles a special case where the objective function is partially separable and hence updating conjugate subspaces is effective for large scale sparse optimization problems. In the article of Chen, Fel and Zheng [2], a "Parallel Quasi-Newton algorithm" is presented that divides the processors in two groups that operate asynchronously and avoids updating the Hessian at every iteration. However the gain, if any, in the case of time consuming objectives is minimal, since the main cost is not in updating an $n \times n$ matrix. Byrd, Schnabel and Shultz [3], discuss a parallelization scheme for the BFGS method based on estimating the gradient and part of the Hessian by finite differencing (which lends itself to extensive parallel computation) and in addition on parallel linear algebra solvers. A similar philosophy is followed for the Newton method, in the work by Conforti and Musmanno [4]. Earlier in [5], Laarhoven presented a method that exploits parallel calls to estimate via updating, (*and not via finite differencing*) the inverse Hessian matrix. His work is based upon a *1973* NASA–report by Straeter [6], that indeed capitalizes on the capabilities offered by parallel processing systems. Phua et al. [7], describe a method where line searches are applied in parallel to several descent directions produced via different Hessian updates. Among the various schemes the SR1, BFGS and the Biggs [8] updates are being considered. An interesting review appeared in 1995 by Schnabel [9], commending on the prospects of parallel non-linear optimization in a broader framework where also the field of global optimization was considered.

In the present article we present optimization algorithms suitable for execution on parallel systems. Our main focus is to accelerate the solution process without much concern for the amount of utilized resources such as the number of processors/cores, memory size, disc space, communication switches etc. One line we pursue is based on a population of M neighbouring points x_i which are used to obtain, via SR1 updating, approximations to the corresponding M Hessians. Next, from each of these M points, a line search is started in the direction determined by:

$$h_i = -B_i^{-1} g_i \equiv -H_i g_i, \quad \forall i = 1, 2, \ldots, M$$

B_i, H_i and g_i denote a modified Hessian, its inverse and the gradient at the point x_i correspondingly. From the new points that emerged after the line–searches, we pick the one with the lowest function value and repeat the process anew; i.e. we pick $M - 1$ additional points in its neighbourhood, estimate the corresponding (M) Hessian matrices and so on so forth, until a termination criterion prevails.

A second approach comes from noticing that trust region methods solve a modified Hessian problem $(B + aI)h = -g$, where a is a parameter determined by the constraint $|h(a)| \leq \rho$, ρ being a proper trust radius controlled externally according to the quality of the quadratic local fit to the objective function.

From the current point we calculate M directions h_i, by picking M values a_i, for the parameter a inside a proper range. In the next step, a line search is started along each direction h_i. From the resulting new points, the best one is selected as the current point. This process is repeated until a convergence criterion is satisfied.

The detailed procedures are described in Sect. 2. Benchmarking experiments have been performed and the results are presented in Sect. 3. Conclusions and directions for further research are contained in Sect. 4.

2 Algorithmic Presentation

In this section we describe in detail the proposed algorithmic schemes. We start with the nomadic search and then we analyse the bundle search. We conclude the section with the presentation of a nested combined scheme.

2.1 Concurrent Nomadic Search

Concurrent nomadic search's main iteration step consists of four basic operations:

- Definition of a nomadic group, i.e. a set of M points $x_1, x_2, \ldots, x_M$.
- Estimation of a positive definite approximation to the corresponding Hessians $B_i \approx \nabla^2 f(x_i),\ i = 1, \ldots, M$.
- Solution of the linear systems: $B_i h_i = -\nabla f(x_i)$, to obtain search directions $h_i,\ i = 1, \ldots, M$.
- Application of M line search procedures to compute the new points as: $x_i + \alpha_i h_i,\ i = 1, \ldots, M$.

The algorithm starts by creating a set of M points $x_1, x_2, \ldots, x_M$ relatively close to each other and randomly chosen in the vicinity of x_1, which is considered to be the starting point. Each point is assigned a Hessian matrix $B_i,\ \ i = 1, \ldots, M$ which is estimated via SR1 updates from the rest $M - 1$ points. The SR1 update formula shown in Eq. (1) has been already used for estimating a Hessian from neighboring points [10,11].

$$B_i' = B_i + \frac{(y - B_i s)(y - B_i s)^\top}{(y - B_i s)^\top s} \tag{1}$$
$$s = x_i - x_j, \quad y = \nabla f(x_i) - \nabla f(x_j)$$

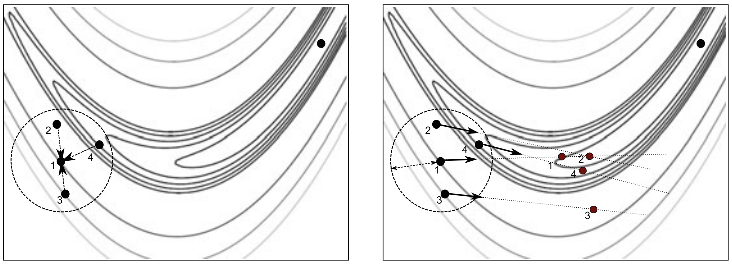
Equation (1) is applied for every point x_i using information communicated by the rest $M - 1$ points. This procedure can be performed concurrently at the extra communication cost of broadcasting location (x_i) and gradient $(\nabla f(x_i))$ information. Since the SR1 update does not maintain positive definiteness, we modify each Hessian matrix using a variant of Choleski decomposition. The Hessian is first decomposed into Choleski LDL^T factors and if the diagonal matrix D contains negative elements, it is modified so as to enforce positive definiteness. The search direction h_i, is determined by replacing the estimated Hessian B_i, with a convex combination $B_i' \equiv (1-\mu_i)B_i+\mu_i I$, with $\mu_i \in [0, 1]$, that creates directions that are Newton–dominant for low values of μ_i and gradient–dominant for high values of μ_i. Note that μ_i is calculated so as to favour a Newton–dominant direction for points with a relatively low objective value, and a gradient–dominant direction for points with a relatively high objective value, as indicated by relation (2).

$$\mu_i = \frac{f(x_i) - F_s}{F_b - F_s}, \quad \forall i = 1, 2, \ldots, M \tag{2}$$

where $F_s = \min f(x_i)$ and $F_b = \max f(x_i)$. The algorithm then performs concurrently M line searches along the h_i directions and computes M new points $x_i + \lambda_i h_i$, where λ_i is the step calculated by the line search procedure. From that point onwards, concurrent nomadic search can either keep these new points and repeat the procedure or maintain the best point and resample $M-1$ points anew (*periodic reset*).

In Fig. 1 we provide a two dimensional illustration of the basic steps and in Algorithm (1) we lay out a detailed description. We consider that the function and gradient evaluations as well as the line searches are the time consuming parts of the algorithm. If M computational resources are available the concurrent execution of the nomadic search can be performed in two steps: (i) computing the function and its gradient and (ii) performing the line search. The line search procedure contains a small (bounded) number of successive function and gradient evaluations.

Nomadic search differs from a Newton method that estimates the Hessian with SR1 updates, in that not only one, but M Hessians are being estimated and M line–search procedures are applied. Since the extra effort is undertaken in parallel, and assuming the availability of M cores, there is no time surcharge. Note also that the Hessians are further transformed via the convex combination with the identity matrix, so that the resulting search directions are properly biased towards the gradient or the Newton direction, as dictated by the respective local values of the objective function. Another important feature of the proposed algorithm is that it does not require $O(n)$ points for approximating the Hessian, in contradistinction to Straeter's approach [6] or to numerical differentiation of the gradient.



(a) Communicating for SR1 updates (b) M line searches in parallel

Fig. 1. Nomadic search algorithm 2-D illustration

Algorithm 1. Nomadic search algorithm

```
Input: Objective function, $f : X \subset \mathbb{R}^n \to \mathbb{R}$; number of points: $M$; , small radius: $R$, a
       small number $\epsilon > 0$
Output: Best detected solution: $x^*$, $f(x^*)$.
1  Select at random $M - 1$ points, $x_2, x_3, \cdots, x_M$, such that
   $|x_1 - x_j| \leq R, \quad \forall j = 2, \cdots, M$.
2  Calculate $f_i = f(x_i), \quad g_i = \nabla f(x_i) \quad \forall i = 1, 2, \cdots, M$ (in parallel)
3  if termination then
       $x^* = x_1 \quad f(x^*) = f_1$
       return
   end
4  for $i = 1, 2, \cdots, M$ do
       Use SR1 updates from all other points $x_j$ with $j \neq i$, to obtain an approximate
       Hessian $B_i \approx \nabla^2 f(x_i)$.
       Decompose (Choleski) $B_i = L_i D_i L_i^T$, and modify $D_i$ so as to render $B_i$ positive
       definite.
   end
5  Calculate $F_s = \min_i \{f_i\}$ and $F_b = \max_i \{f_i\}$
6  Calculate $\mu_i = (1 - \epsilon) \frac{f_i - F_s}{F_b - F_s}, \quad \forall i = 1, 2, \cdots, M$ (in parallel)
7  Solve $[(1 - \mu_i) B_i + \mu_i I]\, h_i = -g_i, \quad \forall i = 1, 2, \cdots, M$ (in parallel)
8  Apply a line–search $\forall i = 1, 2, \cdots, M$ as: (in parallel)
   $\lambda_i^* = \arg\min_\lambda f(x_i + \lambda h_i)$
   Set $f_i = f(x_i + \lambda_i^* h_i)$ (already calculated during the line–search).
9  Set: $x_i \leftarrow x_i + \lambda_i^* h_i, \ \forall i = 1, 2, \cdots, M$ (in parallel)
10 Find the index $k$ for which $f_k = \min_i \{f_i\}, \ \forall i = 1, 2, \cdots, M$
11 Swap $x_1$ and $x_k$
12 if periodic reset then
       Repeat from Step 1
   else
       Repeat from Step 2
   end
```

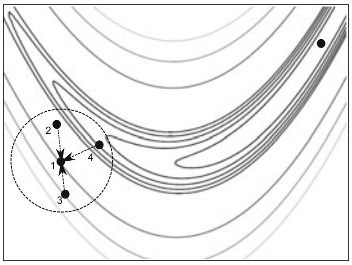
2.2 Bundle Search Algorithm

Bundle search maintains a single point and a set of N descent directions (the bundle) originating from it, along each of which a line–search is to be concurrently applied. The algorithm begins with an estimation of the Hessian matrix, using the same technique as in the case of Nomadic search. The N descent directions are calculated by solving in parallel, N linear systems of the form:

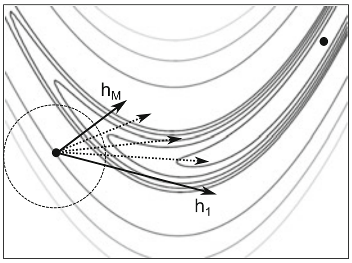
$$[(1 - \mu_i)B + \mu_i I]\, h_i = -g, \quad \forall i = 1, 2, \ldots, N$$

where the quantities μ_i are appropriately chosen in $[0, 1]$ The bundle contains directions that are biased towards the steepest descent for large μ_i and towards the Newton direction for small μ_i. The rationale behind this choice is to exploit the steepest descent if the current point is far from the minimum, and the Newton direction if it is close to it. In Fig. 2 we illustrate on the left side, information communication for Hessian approximation, and on the right side the extend of the bundle. In Fig. 2(b) direction h_1 corresponds to the Newton $(-B^{-1}g)$ while h_N to the steepest descent $(-g)$ direction. In two dimensions the bundle resembles a uniform fan of descent directions between $-g$ and $-B^{-1}g$. In the case of problems of higher dimensionality, the directions of the bundle are not coplanar. After having applied the line searches, the bundle algorithm keeps the point $x' = x + \lambda_J h_J$ with the lowest function value. The next iteration involves random sampling of $M - 1$ points around the kept one and estimation of the new Hessian matrix via $M - 1$ SR1 updates. A complete description of the bundle search is presented in Algorithm 2.

Similar to the nomadic search, bundle search performs two time consuming tasks. Function and gradient calculation for the SR1 updating of the Hessian, and the line searches. Assuming again that we do have N processing units available, a single iteration of the bundle search costs as much as a single function plus gradient evaluation and a line search. The communication costs are reduced in comparison to nomadic search, since in this scheme we have to update only one Hessian matrix.



(a) Communicating for single SR1 update



(b) Following M directions in parallel

Fig. 2. Bundle search 2-D example

2.3 Nested Nomadic and Bundle Search

In order to take advantage of both nomadic and bundle search methodologies, we propose a nested scheme that involves an outer iteration following nomadic search and an inner iteration with bundles of directions. In this scheme we maintain M points $x_i,\ i = 1, \ldots, M$ and perform all–to–all SR1 updates to approximate M Hessian matrices $B_i,\ i = 1, \ldots, M$. From each point we then

Algorithm 2. Bundle search algorithm

```
   Input: Objective function, $f : X \subset \mathbb{R}^n \to \mathbb{R}$; number of directions: $N$; number of points:
          $M$; small radius: $R$, a small number $\epsilon > 0$
   Output: Best detected solution: $x^*$, $f(x^*)$.
1  if termination then
       $x^* = x \quad f(x^*) = f$
       return
   end
2  Decompose (Choleski) $B = LDL^T$, and modify $D$ so as to render $B$ positive definite.
3  Solve $[(1-\mu_i)B + \mu_i I]\, h_i = -g, \quad \forall i = 1, 2, \cdots, N$ (in parallel)
   where $\mu_i \in [0, 1], \ \forall i = 1, \cdots, N$.
4  Apply a line–search $\forall i = 1, 2, \cdots, N$ as: (in parallel)
   $\lambda_i^* = \arg\min_{\lambda} f(x + \lambda h_i)$
   Set $f_i = f(x + \lambda_i^* h_i)$ (already calculated during the line–search).
5  Find the index $k$ for which $f_k = \min_i \{f_i\}, \ \forall i = 1, 2, \cdots, N$
6  Set: $x \leftarrow x + \lambda_k^* h_k$.
7  Set $f \leftarrow f_k$, and $g \leftarrow \nabla f(x_k)$
8  Select at random $M-1$ points, $x_2, x_3, \cdots, x_M$, such that
   $|x - x_j| \le R, \quad \forall j = 2, \cdots, M$.
9  Calculate the Hessian approximation at $x$ via SR1 using the adjacent points
   $x_2, x_3, \cdots, x_M$.
10 Go to 1
```

Algorithm 3. Nested nomadic and bundle scheme

```
   Input: Objective function, $f : X \subset \mathbb{R}^n \to \mathbb{R}$; number of points: $M$; number of directions
          $N$ small radius: $R$, a small number $\epsilon > 0$
   Output: Best detected solution: $x^*$, $f(x^*)$.
1  Select at random $M-1$ points, $x_2, x_3, \cdots, x_M$, such that
   $|x_1 - x_j| \le R, \quad \forall j = 2, \cdots, M$.
2  Calculate $f_i = f(x_i), \quad g_i = \nabla f(x_i) \quad \forall i = 1, 2, \cdots, M$ (in parallel)
3  if termination then
       $x^* = x_1 \quad f(x^*) = f_1$
       return
   end
4  Estimate $B_i \approx \nabla^2 f(x_i)$ and decompose so as to render it positive definite
   $\forall i = 1, 2, \cdots, M$ (in parallel)
5  Calculate $F_s = \min_i \{f_i\}$ and $F_b = \max_i \{f_i\}$ needed in $\mu_i = (1-\epsilon)\dfrac{f_i - F_s}{F_b - F_s}$
6  Solve $[(1-\mu_i)B_i + \mu_i I]\, h_i = -g_i, \quad \forall i = 1, 2, \cdots, M$ (in parallel)
   // For all points
   for $i = 1, 2, \ldots, M$ do
       // Begin bundle search
       for $j = 1, 2, \ldots, N$ do
7          Solve $\left[(1-\tilde{\mu}_j)B_i + \tilde{\mu}_j I\right] h_j = -g_i$, where $\tilde{\mu}_j = (1-\epsilon)\dfrac{j-1}{N-1}$.
8          Apply a line–search $\lambda_j^* = \arg\min_{\lambda} f(x_i + \lambda h_j)$
           Set $\tilde{f}_j = f(x_i + \lambda_j^* h_j)$.
       end
9      Find the index $k$ for which $f_k = \min_j \{\tilde{f}_j\}, \ \forall j = 1, 2, \cdots, M$
10     Set: $x_i \leftarrow x_i + \lambda_k^* h_k, \quad f_i \leftarrow f_k$, and $g_i \leftarrow \nabla f(x_k)$.
   end
11 Find the index $k$ for which $f_k = \min_i \{f_i\}, \ \forall i = 1, 2, \cdots, M$
12 Swap $x_1$ and $x_k$
13 if fresh restart then
       Repeat from Step 1
   else
       Repeat from Step 2
   end
```

define N directions by following the bundle search methodology. The points $x_i, \ i = 1, \ldots, M$ of the next iteration are taken from the results of the N line searches. In this nested scheme we define $M \times N$ line search tasks in parallel (Step 8 of Algorithm 3). In Algorithm 3 we present the nested scheme.

The nested scheme is even more demanding on computational resources, but we expect to further reduce the length of execution times. Efficient implementation of nested schemes need advanced runtime support [12,13] which is currently available and supported by OpenMP API.

3 Numerical Experiments

We have implemented all parallel algorithms in Matlab in order to measure their effectiveness with respect to the number of iterations. It is not a true par-

Table 1. Speedup in iterations for $M = 8, 16$

		MP (8)		MD(8)		Nested (4×2)		MP (16)		MD (16)		Nested (4×4)	
		SR1	BFGS	SR1	BFGS	SR1	BFGS	SR1	BFGS	SR1	BFGS	SR1	BFGS
Heli	3	1.40	1.80	1.75	2.25	1.11	1.42	1.62	2.08	1.50	1.93	1.40	1.80
Gaussian	3	1.86	1.71	1.63	1.50	1.86	1.71	2.17	2.00	2.17	2.00	2.17	2.00
Var Dim.	2	2.00	2.00	2.00	2.00	1.60	1.60	2.00	2.00	2.00	2.00	2.00	2.00
Watson	2	1.60	1.80	1.60	1.80	1.60	1.80	1.60	1.80	1.60	1.80	1.60	1.80
Brown	4	16.00	55.56	24.00	83.33	12.00	41.67	16.00	55.56	18.00	62.50	16.00	55.56
Gulf	3	5.81	1.33	5.55	1.27	6.78	1.56	6.42	1.47	7.63	1.75	7.18	1.65
Trigon	2	1.67	1.17	2.00	1.40	1.67	1.17	2.00	1.40	2.00	1.40	2.00	1.40
Rosen.	2	3.00	2.54	2.44	2.06	1.44	1.22	2.05	1.74	4.33	3.67	2.05	1.74
Beale	2	1.44	1.56	1.63	1.75	0.57	0.61	1.86	2.00	1.86	2.00	0.65	0.70
Wood	4	0.59	0.68	0.79	0.89	0.71	0.81	0.73	0.83	0.81	0.93	0.92	1.04
Cheby.	2	1.50	1.50	1.20	1.20	0.86	0.86	1.50	1.50	2.00	2.00	1.00	1.00
Cubic	2	3.48	1.30	3.64	1.36	1.48	0.56	3.64	1.36	5.33	2.00	1.18	0.44
De Jong1	3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
De Jong2	2	1.50	1.31	1.41	1.24	0.65	0.57	1.50	1.31	1.50	1.31	0.67	0.58
Goldstein	2	1.33	1.44	2.00	2.17	1.33	1.44	1.71	1.86	2.00	2.17	1.71	1.86
Branin	2	1.00	1.00	2.00	2.00	1.00	1.00	1.20	1.20	1.50	1.50	1.00	1.00
Shekel5	4	2.83	2.50	3.40	3.00	2.83	2.50	2.83	2.50	3.40	3.00	3.40	3.00
Shekel10	4	1.50	2.00	1.80	2.40	1.29	1.71	1.50	2.00	1.80	2.40	1.50	2.00
Six hump	2	6.00	1.60	7.50	2.00	4.29	1.14	6.00	1.60	7.50	2.00	5.00	1.33
Colville	4	1.57	2.07	1.29	1.71	0.69	0.91	1.22	1.61	1.47	1.93	1.10	1.45
Bazaraa	2	5.50	4.50	2.20	1.80	1.69	1.38	1.83	1.50	2.20	1.80	7.33	6.00
Quadratic	2	3.00	9.00	3.00	9.00	3.00	9.00	3.00	9.00	3.00	9.00	3.00	9.00
Var Dim.	10	2.78	2.78	2.78	2.78	3.13	3.13	2.78	2.78	3.57	3.57	3.13	3.13
Watson	10	2.45	2.20	1.07	0.96	1.75	1.57	3.27	2.93	1.11	1.00	2.58	2.32
Trigon	10	2.80	3.93	3.23	4.54	3.23	4.54	4.67	6.56	3.82	5.36	3.50	4.92
Rosen.	10	4.86	1.69	8.95	3.11	3.94	1.37	7.88	2.74	10.94	3.81	5.12	1.78
Quadratic	10	20.00	8.67	30.00	13.00	12.00	5.20	60.00	26.00	60.00	26.00	20.00	8.67
Var Dim.	50	2.09	0.36	1.44	0.25	1.28	0.22	1.64	0.29	2.09	0.36	1.92	0.33
Watson	50	1.45	0.90	0.47	0.29	0.74	0.46	2.44	1.51	0.34	0.21	0.95	0.59
Trigon	50	2.24	2.55	3.70	4.22	2.74	3.13	3.27	3.73	3.70	4.22	4.72	5.39
Rosen.	50	1.57	0.91	1.02	0.59	1.19	0.69	2.14	1.23	2.24	1.29	1.89	1.09
Quadratic	50	15.73	7.82	15.73	7.82	6.92	3.44	28.83	14.33	43.25	21.50	12.36	6.14
Var Dim.	100	1.57	43.48	1.89	52.63	0.92	25.64	1.29	35.71	2.77	76.92	2.57	71.43
Watson	100	1.32	7.35	0.39	2.16	0.66	3.69	1.54	8.55	0.86	4.78	0.98	5.46
Trigon	100	7.93	1.77	4.79	1.07	9.49	2.12	8.34	1.86	9.68	2.16	9.88	2.20
Rosen.	100	1.84	0.81	1.00	0.44	2.00	0.88	2.40	1.05	1.53	0.67	2.02	0.88
Quadratic	100	7.90	6.52	5.03	4.15	2.44	2.01	15.09	12.45	18.44	15.22	5.03	4.15
Speedup:		**3.84**	**5.17**	**4.20**	**6.08**	**2.75**	**3.61**	**5.65**	**5.92**	**6.46**	**7.49**	**3.80**	**5.86**

allel implementation, e.g. using Matlab's parallel toolbox, but it is used as a proof of concept for the efficiency of the nomadic, bundle and the combined search. The comparison is made against two well known quasi–Newton sequential algorithms: BFGS and SR1, each with a line search. All methods in the comparison table share the same line search code. The basic computational cost of these methods per iteration is one function and gradient evaluation and one line search. Considering the communication costs negligible with respect to function/gradient evaluation, it is plausible to claim that one iteration of a sequential quasi–Newton algorithm and that of our proposed parallel methodologies, take the same amount of time. Hence in this study we compare the number of iterations to provide a proof–of–concept, expecting that the estimated speedup is close to that of a real parallel implementation.

We used a part of the well established Moré optimization test functions [14] and some instances from the Dixon-Szego test set. For every test function we report the number of iterations each algorithm performed in order to reach the target minimizer starting from a pre-specified point. All numbers reported are averages of twenty runs with different random seeds. We have experimented with values of $M = 8, 16, 48, 64$. In Table 1 we present relative speedups, with respect to serial BFGS and DFP methods, up for the cases $M = 8, 16$ and in Table 2 for the cases $M = 48, 64$. In the last row we present the average speedup for all test functions.

By inspecting the result tables we can see that the proposed parallel algorithms can result in 6 times less iterations than their serial competitors which represent the state–of–the–art in the field on numerical optimization. A closer look reveals that in some cases (eg. Quadratic 50, 100, Rosenbrock 10, Brown and Dennis, Trigonometric) the speed up in terms of iterations is noteworthy when M is greater than 16. These results indicate that with a proper implementation and a sufficiently heavy objective function evaluation, nomadic, bundle and nested concurrent searches may be used to accelerate convergence by a substantial factor. Is obvious though that the speedup does not scale well with M. This can be attributed to the fact that near the minimum all directions tend to coincide with the Newton direction, hence in these last iterations the alternatives offer almost no advantage. Dynamic allocation of computational resources and batch optimization schemes may increase the overall ratio.

4 Conclusions

We have presented three parallel methods for the problem of local optimization with line searches. A multipoint or concurrent nomadic search, a multi–direction or concurrent bundle search, and a combination of the two. All use SR1 updates from randomly sampled points to estimate required Hessian matrices. Preliminary simulation results clearly indicate that they may significantly reduce the number of iterations, and consequently the overall computational time, needed by well established and widely used serial rival methods.

For the nomadic search algorithm the selection of points for the next iteration is an issue that needs further examination. This is a practical, yet important

Table 2. Speedup in iterations for $M = 48, 64$

		MP (48)		MD (48)		Nested (12×4)		MP (64)		MD (64)		Nested (16×4)	
		SR1	BFGS	SR1	BFGS	SR1	BFGS	SR1	BFGS	SR1	BFGS	SR1	BFGS
Heli	3	1.62	2.08	1.75	2.25	1.91	2.45	1.62	2.08	1.75	2.25	1.91	2.45
Gauss	3	2.17	2.00	1.86	1.71	2.17	2.00	2.17	2.00	2.17	2.00	2.17	2.00
VarD	2	2.00	2.00	2.67	2.67	2.00	2.00	2.00	2.00	2.67	2.67	2.67	2.67
Wats	2	1.60	1.80	2.67	3.00	1.60	1.80	1.60	1.80	2.00	2.25	2.00	2.25
Brow	4	16.00	55.56	20.57	71.43	18.00	62.50	16.00	55.56	18.00	62.50	18.00	62.50
Gulf	3	7.18	1.65	10.17	2.33	8.13	1.87	7.63	1.75	12.20	2.80	8.13	1.87
Trigon	2	2.00	1.40	1.67	1.17	2.00	1.40	2.00	1.40	2.00	1.40	2.00	1.40
Rosen.	2	3.90	3.30	3.00	2.54	1.77	1.50	3.55	3.00	3.25	2.75	2.05	1.74
Beale	2	1.86	2.00	1.63	1.75	1.18	1.27	2.17	2.33	1.63	1.75	1.00	1.08
Wood	4	0.88	1.00	0.81	0.93	1.22	1.39	0.96	1.09	0.88	1.00	1.38	1.56
Cheby.	2	1.50	1.50	1.50	1.50	1.00	1.00	1.50	1.50	1.50	1.50	1.00	1.00
Cubic	2	3.81	1.43	5.33	2.00	3.81	1.43	4.44	1.67	5.33	2.00	3.64	1.36
Jong1	3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Jong2	2	1.60	1.40	1.85	1.62	1.60	1.40	1.60	1.40	1.85	1.62	0.89	0.78
Gold	2	2.00	2.17	2.40	2.60	1.71	1.86	1.71	1.86	2.40	2.60	2.00	2.17
Branin	2	1.20	1.20	1.50	1.50	1.20	1.20	1.20	1.20	1.50	1.50	1.20	1.20
S5	4	2.83	2.50	3.40	3.00	3.40	3.00	2.83	2.50	3.40	3.00	3.40	3.00
S10	4	1.50	2.00	1.80	2.40	1.50	2.00	1.50	2.00	1.80	2.40	1.50	2.00
Hump	2	6.00	1.60	10.00	2.67	5.00	1.33	6.00	1.60	7.50	2.00	5.00	1.33
Colv	4	1.57	2.07	1.38	1.81	1.16	1.53	1.57	2.07	1.38	1.81	1.29	1.71
Baza	2	2.75	2.25	2.20	1.80	5.50	4.50	3.67	3.00	2.20	1.80	4.40	3.60
Quad	2	3.00	9.00	3.00	9.00	3.00	9.00	3.00	9.00	3.00	9.00	3.00	9.00
VarD	10	4.17	4.17	4.17	4.17	3.57	3.57	4.17	4.17	4.17	4.17	3.57	3.57
Wats	10	3.50	3.14	1.58	1.42	3.27	2.93	3.50	3.14	1.48	1.33	3.50	3.14
Trigon	10	4.20	5.90	10.50	14.75	5.25	7.38	3.82	5.36	4.67	6.56	6.00	8.43
Rose	10	8.57	2.98	12.31	4.28	9.61	3.34	7.43	2.58	12.71	4.42	10.65	3.70
Quad	10	60.00	26.00	60.00	26.00	60.00	26.00	60.00	26.00	60.00	26.00	60.00	26.00
VarD	50	1.92	0.33	1.92	0.33	2.30	0.40	2.09	0.36	1.92	0.33	2.30	0.40
Wats	50	3.45	2.14	0.83	0.51	2.22	1.38	4.00	2.48	1.16	0.72	2.63	1.63
Trigon	50	4.05	4.62	3.40	3.88	5.00	5.71	3.15	3.59	3.54	4.04	4.72	5.39
Rose	50	3.36	1.93	9.09	5.24	3.33	1.92	5.38	3.10	9.26	5.33	3.70	2.13
Quad	50	57.67	28.67	86.50	43.00	28.83	14.33	173.00	86.00	173.00	86.00	34.60	17.20
VarD	100	1.71	47.62	1.50	41.67	2.57	71.43	1.71	47.62	1.33	37.04	2.57	71.43
Wats	100	1.08	6.02	1.31	7.30	2.54	14.08	1.29	7.19	1.43	7.94	2.17	12.05
Trigon	100	11.52	2.57	20.17	4.50	20.17	4.50	13.83	3.09	16.13	3.60	16.69	3.72
Rose	100	3.97	1.74	6.33	2.77	4.18	1.83	4.03	1.77	11.90	5.21	4.50	1.97
Quad	100	33.20	27.40	55.33	45.67	15.09	12.45	41.50	34.25	55.33	45.67	20.75	17.13
Speedup:		**7.31**	**7.19**	**9.65**	**8.81**	**6.43**	**7.53**	**10.77**	**8.99**	**11.82**	**9.46**	**6.70**	**7.72**

issue, since it may affect seriously the performance of the method. For the Hessian of the maintained point (or points) one may consider to either discard the existing information and proceed as in the initialization step using the SR1 scheme, or to continue updating the existing Hessian. Another question that may arise in the second case, is whether a periodic reset is then necessary and at what frequency. In addition, the Hessian may also be updated right after the line search application, for instance via a BFGS or any other Quasi-Newton formula. It is a matter of further investigation if this will enhance the methods performance or not. Most Hessian issues referred to above, need to be investigated for the case of the bundle search as well. We intend first to implement these methods in a real parallel system (MPI multicore cluster) and then we would like to address the above important issues.

Acknowledgments. This work is co-financed by the European Union and Greece Operational Program "Human Resources Development" -NSFR 2007–2013 - European Social Fund.

References

1. Chen, M.-Q., Han, S.-P.: A parallel quasi-Newton method for partially separable large scale minimization. Ann. Oper. Res. **14**, 195–211 (1998)
2. Chen, Z., Fel, P., Zheng, H.: A parallel quasi-Newton algorithm for unconstraint optimization. Computing **55**, 125–133 (1995)
3. Byrd, R.H., Schnabel, R.B., Shultz, G.A.: Parallel quasi-Newton methods for unconstrained optimization. Math. Program. **42**, 273–306 (1988)
4. Conforti, D., Musmanno, R.: A parallel asynchronous Newton algorithm for unconstrained optimization. J. Optim. Theor. Appl. **77**, 305–322 (1993)
5. van Laarhoven, P.J.M.: Parallel variable metric algorithms for unconstrained optimization. Math. Program. **33**, 68–81 (1985)
6. Straeter, T.A.: A parallel variable metric optimization algorithm. NASA Technical Note D-7329, Hampton, VA (1973)
7. Phua, P.K.H., Fan, W., Zeng, Y.: Self-scaling parallel quasi-Newton methods. In: Fourth International Conference on Optimization: Techniques and Applications, Australia (1998)
8. Biggs, M.C.: A note on minimization algorithms which make use of non-quadratic properties of the objective function. J. Inst. Math. Appl. **12**, 337–338 (1973)
9. Schnabel, R.B.: A view of the limitations, opportunities, and challenges in parallel nonlinear optimization. Parallel Comput. **21**, 875–905 (1995)
10. Fayez Khalfan, H., Byrd, R.H., Schnabel, R.B.: A theoretical and experimental study of the symmetric rank-one update. SIAM J. Optim. **3**(1), 1–24 (1993)
11. Tu, W., Mayne, R.W.: Studies of multi-start clustering for global optimization. Int. J. Numer. Meth. Eng. **53**(9), 2239–2252 (2002)
12. Voglis, C., Hadjidoukas, P.E., Dimakopoulos, V.V., Lagaris, I.E., Papageorgiou, D.G.: Task-parallel global optimization with application to protein folding. In: High Performance Computing and Simulation (HPCS), pp. 186–192. IEEE (2011)
13. Narang, A., Srivastava, A., Katta, N.P.K.: Distributed scalable collaborative filtering algorithm. In: Jeannot, E., Namyst, R., Roman, J. (eds.) Euro-Par 2011, Part I. LNCS, vol. 6852, pp. 353–365. Springer, Heidelberg (2011)
14. Moré, J.J., Garbow, B.S., Hillstrom, K.E.: Testing unconstrained optimization software. ACM Trans. Math. Softw. (TOMS) **7**(1), 17–41 (1981)

Parallel Multi-objective Memetic Algorithm for Competitive Facility Location

Algirdas Lančinskas[✉] and Julius Žilinskas

Institute of Mathematics and Informatics, Vilnius University,
Akademijos 4, 08663 Vilnius, Lithuania
{algirdas.lancinskas,julius.zilinskas}@mii.vu.lt
http://www.mii.vu.lt

Abstract. A hybrid genetic algorithm for global multi-objective optimization is parallelized and applied to solve competitive facility location problems. The impact of usage of the local search on the performance of the parallel algorithm has been investigated. An asynchronous version of the parallel genetic algorithm with the local search has been proposed and investigated by solving competitive facility location problem utilizing hybrid distributed and shared memory parallel programming model on high performance computing system.

Keywords: Facility location · Multi-objective optimization · Memetic algorithms

1 Introduction

The location of facilities is important for firms that provide goods or services to customers in a certain geographical area. There are a lot of facility location models proposed in literature [1–3]. They vary on different properties such as location space, which can be continuous or discrete, customer behavior, function of facility attraction, etc. Most of the models deal with Competitive Facility Location (CFL), where a new firm wants to enter the market. The *entering firm* competes with other firms already in the market, with respect to maximize the market share or profit, taking into account the behavior of customers in the region of interest. Some variants of models of behavior of customers have been proposed by Huff [4], where attraction that particular customer feels to the facility is measured by the ratio of the quality value of the facility and the distance between the facility and the customer.

Another case in CFL is *firm expansion*. Here firm already in the market is interested in increasing its market share by establishing a set of new facilities. The expanding firm is aimed at maximization of the market share of the new facilities, however, the new facilities can attract customers that already served by other facilities belonging to the expanding firm, thus causing the effect of *cannibalism*. Thus the expanding firm faces a multi-objective optimization problem with the following two objectives: (1) to maximize market share of the new facilities, and (2) minimize the effect of cannibalism.

2 Multi-objective Optimization

Usually it is impossible to find the solution which would be the best according to many objectives – the best solution by one objective could be not the best or even the worst by another one.

In terms of multi-objective optimization, it is said that solution $\mathbf{x}_1$ *dominates* solution $\mathbf{x}_2$ if and only if (1) solution $\mathbf{x}_1$ is not worse than solution $\mathbf{x}_2$ by all objectives, and (2) solution $\mathbf{x}_1$ is strictly better than solution $\mathbf{x}_2$ by at least one objective.

Such a dominance relation is denoted by $\mathbf{x}_1 \succ \mathbf{x}_2$, and solution $\mathbf{x}_1$ is called *dominator* of $\mathbf{x}_2$. The number of dominators of the solution $\mathbf{x}_2$ is called *non-dominance rank* (or simply *Pareto rank*) of $\mathbf{x}_2$, and the solution which has no dominators is called *non-dominated* or *Pareto-optimal*. The set of non-dominated solutions is called *Pareto set*, and the set of corresponding values of objective functions is called *Pareto front*.

Finding the exact Pareto front is usually difficult and time consuming, or even impracticable task. Therefore algorithms, providing an approximation of the true Pareto front are often used. Recently evolutionary algorithms (EAs) became very popular for estimation of the solution of various practical optimization problems, including the multi-objective ones. These algorithms are easy to implement, require little knowledge about the problem being solved, and are well suited to parallel computing. Multi-Objective EAs (MOEAs) can yield a whole set of potential solutions, which are all optimal in some sense, and give the option to assess the trade-offs between different solutions. Different MOEAs have been proposed in the literature (see [5]), most popular of which are Vector Evaluated Genetic Algorithm (VEGA) [6], Strength Pareto Evolutionary Algorithm (SPEA and SPEA2) [7,8], Pareto Archived Evolutionary Strategy (PAES) [9], and Non-dominated Sorting Genetic Algorithm (NSGA, NSGA-II) [10,11].

3 Memetic Algorithm Based on NSGA-II

The term *memetic* was introduced by Moscato [12], and refers to hybridization of evolutionary or any population-based approach with a local search. In this paper we consider a memetic algorithm, derived by incorporation of a local search into Non-dominated Sorting Genetic Algorithm.

3.1 Non-dominated Sorting Genetic Algorithm

Non-dominated Sorting Genetic Algorithm (NSGA) was firstly proposed by Srinivas and Deb [10], and was one of the first multi-objective optimization algorithms applied to various problems [13,14]. The updated version of the algorithm, NSGA-II, was proposed by Deb et al. [11].

The algorithm begins with an initial set P_1, called *parent* population, consisting of N candidate solutions (*individuals*), randomly generated over the feasible area. A new *child* population Q_1 of the same size as the parent population is generated by applying genetic operators (selection, crossover, and mutation) to the individuals of the parent population. Both populations are merged into one $2N$-size population R_1, whose individuals are evaluated by the dominance relation between each other. A new parent population P_2 is formed by selecting N least dominated individuals from R_1. If two or more individuals are equally dominated, then a crowding distance estimator [15] is used to choose the most promising one. A population P_2 is then used as a parent population to generate a child population Q_2 in the next iteration, called *generation*. Such an iterative process is continued till the stopping criterion is satisfied. The stopping criterion usually is based on the number of generations or function evaluations.

3.2 Memetic Algorithm

In this section we will describe a memetic algorithm derived by hybridizing NSGA-II and Multi-Objective Single Agent Stochastic Search (MOSASS).

The local search MOSASS begins with an initial solution x and returns a set A of non-dominated solutions. A candidate solution x' is generated by adding a random vector ξ, generated utilizing Gaussian perturbations: $x' = x + \xi$. If $x' \succ x$ then x is updated by x'; if $x' \not\prec x$, but is not dominated by any solution in $A \cup \{x\}$, then A is supplemented by x', thus forming the set of non-dominated solutions. If neither of the latter conditions are satisfied, then an opposite candidate solution $x'' = x - \xi$ is considered. The mean and the standard deviation of the Gaussian perturbation are dynamically adjusted with respect to the repetitive successes and failures in generating a candidate solution.

The concept of MOSASS and its incorporation into NSGA-II have been introduced in [16]. An updated version of a hybrid algorithm, called NSGA/LSP, was developed in [17], where probabilistic generation of a neighbor candidate solution in the local search, called MOSASS/P, has been applied.

NSGA/LSP begins with performing NSGA-II generations as described in Sect. 3.1. If the number G of NSGA-II generations performed since the last local search (or the start of the algorithm if the local search has not yet performed) exceeds the predefined value, then the local search is initiated. An auxiliary set P_L of the predefined number k of individuals is created from the population P_g, where g stands for the number of generations, performed since the start of the algorithm. The set P_L is formed by choosing all non-dominated individuals, and removing $|P_L| - k$ randomly selected ones, if $|P_L| > k$; in a case of $|P_L| < k$, the set P_L is supplemented by $k - |P_L|$ individuals, randomly selected from the dominated ones.

Each individual in P_L is locally optimized by MOSASS/P using the predefined budget E_L of evaluations of the objective function. The local optimization of a single individual returns a set of individuals which are non-dominated between each other (for more details we refer to [17]). Since the set P_L of individuals are locally optimized, the whole local optimization returns $|P_L|$ sets of newly generated individuals. All these sets are combined into one set R_g together with the parent population P_g. Individuals of the set R_g are then evaluated with respect to the dominance relation, and the set is reduced to the size of N by removing the most dominated individuals. The algorithm is continued by resetting the counter G to 0, and performing regular NSGA-II generations using the reduced set R as the parent population P_{g+1}.

4 Parallel Memetic Algorithm

Although NSGA-II is known as fast, it still requires a lot of computational resources solving complex optimization problems. There are some works related to the parallelization of NSGA-II [18–20], however most of them are based on parallelization of function evaluations. Another part of the algorithm causing the bottleneck effect in parallel computing is the Pareto ranking of the individuals. Several strategies to utilize parallel computing for Pareto ranking have been proposed in [21], and investigated using a large scale High Performance Computing system in [22].

This research is aimed on parallelization of the memetic algorithm for multi-objective optimization, described in Sect. 3.2. In contrast with parallel version of NSGA-II, proposed in [22], NSGA/LSP has additional part, devoted for the local search, which needs to be parallelized. Although MOSASS/P is iterative and the local optimization of a single individual is considered as a sequential part of the algorithm, the optimization of different individuals can be considered as independent tasks and assigned to different computing resources. On the other hand, the local optimization of a set P_L of individuals returns $|P_L|$ sets, each of which consists of up to E_L newly generated individuals, thus possibly causing the bottleneck effect when managing the sets.

The developed parallel memetic algorithm, is based on the master-slave strategy where all processors have equal computational workload, except the master one, which additionally has to form a new parent population and is responsible for communication between processors. The algorithm, called ParNSGA/LSP, is based on the following process:

(1) The master processor generates the initial parent population P_1 and distributes it among all slaves.
(2) Each processor (including the master) evaluates objective functions of the corresponding subset $P_1^i \subset P_1$ of individuals, where $i = 1, 2, \dots, p$ stands for the ID of the processor; p – the total number of processors; $|P_1^i|$ equals to N/p. The master then gathers the information about objective values and sets the generation counter $g = 1$.
(3) The master processor distributes the parent population P_g among the slaves.
(4) The algorithm behaves depending on the value of g indicating the number of the current generation.
    - If the value of g is not divisible by the predefined number E_L, then a regular NSGA-II generation is performed. Each processor generates an appropriate part Q_g^i of the child population Q_g, and evaluates objective functions values of the newly generated individuals;

- If the value of g is divisible by E_L then the local search is initiated. The master processor forms the set P_L of individuals to be locally optimized, and distributes the formed set among the slaves. Each processor performs local optimization of the corresponding subset P_L^i of individuals, thus obtaining $|P_L^i|$ sets of new individuals. All the obtained sets are combined into one set Q_g^i.

(5) The master processor gathers the subpopulations Q_g^i from the slaves, combines them into one child population Q_g, and distributes it among the slaves.

(6) Each processor combines the received child population Q_g together with the parent population P_g into one population $R_g = Q_g \cup P_g$, and evaluates the Pareto ranks of each $i + j \cdot p$-th individual in R_g, where i stands for the ID of the processor, $j = 1, 2, \ldots$, and $i + j \cdot p \leq |R_g|$.

(7) The master processor gathers all information about the Pareto ranks, counts the total values, and distributes them among the slaves.

(8) All processors continue with forming a new parent population P_{g+1} by selecting N least dominated solutions from R_g, increasing the generations counter g by one, and proceeding to the next generation (4-th step).

If the number of processors is larger than the number of solutions, selected for the local search ($p > |P_L|$), then $|P_L - k|$ processors have to be idle while the local optimization is being performed. In order to avoid such an idle time of some processors, an asynchronous version of the parallel algorithm has been developed and denoted by AsyncParNSGA/LSP. The algorithm is based on assignment of the appropriate number of regular NSGA-II generations for processors which are idle during the local search. The number of the generations to be performed in parallel with the local search is chosen with respect to perform the same number of evaluations of objective functions as during the local search.

Distribution of the information among the slave processors as well as gathering of the information from the slaves is performed following the hierarchic fashion as illustrated in Fig. 1. Communication between processors is performed using Message Passing Interface (MPI) libraries. Although the illustrated strategy for distribution/gathering of the information is quite fast, and communication costs are insignificant using several or several tens of processors, it still can be time consuming using several hundreds or thousands of processors.

In order to reduce the costs of communication between the processors, a hybrid shared-distributed memory parallel programming model (MPI-OpenMP) has been utilized. The processors are grouped into p_1 groups each of which consists of p_2 shared memory processors, where $p_1 \cdot p_2 = p$. In each of the groups processors communicate through the shared memory using OpenMP, whereas communication between the groups is performed using MPI. In contrast with the usage of MPI alone, where information can be distributed/gathered in $\log_2 p$ operations of MPI, using MPI-OpenMP distribution/gathering of the information can be performed using $\log_2 p_1$ MPI operations, where $p_1 < p$.

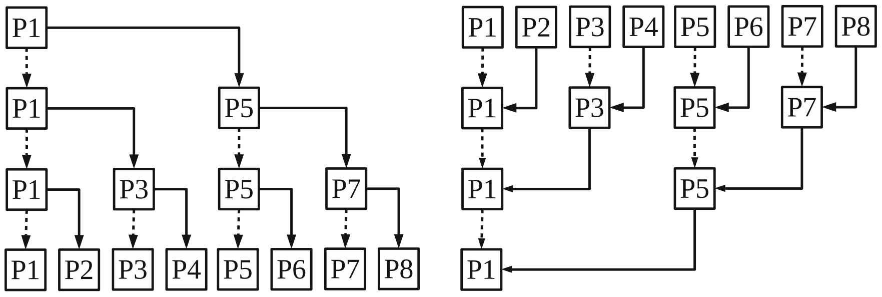


Fig. 1. Illustration of distribution of the information among (left), and gathering from (right) the processors, using distributed memory parallel programming model.

5 Numerical Experiments

Parallel algorithm ParNSGA/LSP has been applied to solve CFL using the High Performance Computing (HPC) system HECToR (High End Computing Terascale Resource) at Edinburgh Parallel Computing Centre (EPCC).

It was expected to locate 5 facilities thus defining the number of variables equal to 10 (each facility has 2 coordinates). Real data with geographical coordinates and populations of around 12000 cities and villages in Lithuania has been used.

The algorithm has been run for 256000 function evaluation using population of 1024 individuals (250 generations of NSGA-II). The local search has been performed after every 10240 (10 generations of NSGA-II) function evaluations for 512 best solutions. Two values of the parameter E_L defining the number of function evaluations devoted for the local search have been investigated: $E_L = 10240$ and $E_L = 20480$.

Computations have been performed using up to 128 nodes containing 16 AMD Opteron 2.3 GHz shared memory processors per each node (2048 processors in total). Performance of the algorithms has been measured by the speed-up of the algorithm

$$S_p = \frac{T_0}{T_p}, \tag{1}$$

where T_0 stands for duration of the sequential algorithm, and T_p – for duration of the parallel algorithm using p processors.

The performance of ParNSGA/LSP has been compared with the performance of parallel version of the classical NSGA-II algorithm, denoted by ParNSGA, parallelized under the same strategy as ParNSGA/LSP.

Results, obtained using ParNSGA and ParNSGA/LSP with $E_L = 10240$ and $E_L = 20480$ (10 and 20 generations, respectively) are presented in Fig. 2, where the horizontal axis represents the number of processors, and the vertical axis – the speed-up of the algorithm. The results show that the algorithm with the local search has a better speed-up if the number of processors is less than or equal

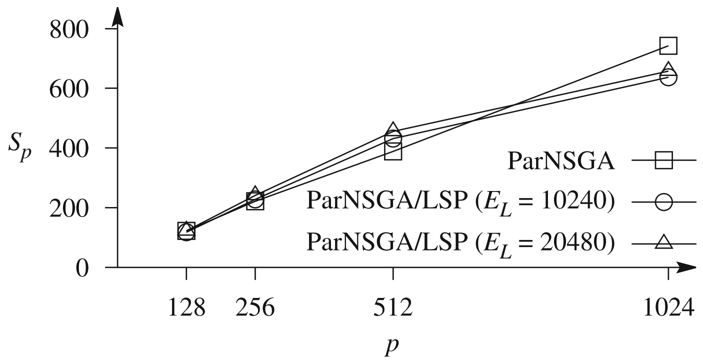


Fig. 2. Speed-up of ParNSGA and ParNSGA/LSP using different E_L values

to 512. We can also see, that memetic algorithm, which devotes more function evaluations for the local search, has a better speed-up also.

When the number of processors is larger than the number of solutions selected for the local optimization, some processors became idle while the local search is being performed. Therefore, the performance of the algorithm ParNSGA/LSP when using 1024 processors is worse than the performance of ParNSGA which does not use the local search. In order to avoid such an idle time of some processors, an asynchronous version of parallel algorithm AsyncParNSGA/LSP with $E_L = 20480$ has been investigated. The comparison of AsyncParNSGA/LSP versus ParNSGA/LSP is shown in Fig. 3. From the figure we can see that devotion of NSGA generations for idle processors significantly increases the performance of the parallel algorithm.

Since the algorithm ParNSGA/LSP is synchronous, it has the same behaviour as the sequential NSGA/LSP. Therefore, the precision of the algorithms must be the same. The same applies to the precision of AsyncParNSGA/LSP if $p \leq |P_L|$. In a case of $p > |P_L|$ the behavior of the parallel algorithm differs from the behavior of the sequential one, thus possibly changing the precision of the approximation. The investigation of the impact of the asynchrony showed that the average precision of the approximation obtained using AsyncParNSGA/LSP was less than 2 % lower than the precision of the approximation obtained by ParNSGA/LSP.

Third experiment has been devoted to investigate the impact of utilization of hybrid distributed-shared memory parallel computing model. All processors have been grouped into groups of 16 shared memory processors. Processors within a group communicate with each other through shared memory (using OpenMP), and communication between groups has been performed by passing messages (using Message Passing Interface, MPI). The respective parallel algorithms have been denoted by AsyncParNSGA/LSP (MPI) and AsyncParNSGA/LSP (MPI-OMP). The local search has been performed after 20480 function evaluations. The obtained results, presented in Fig. 4, show that utilization of hybrid mem-

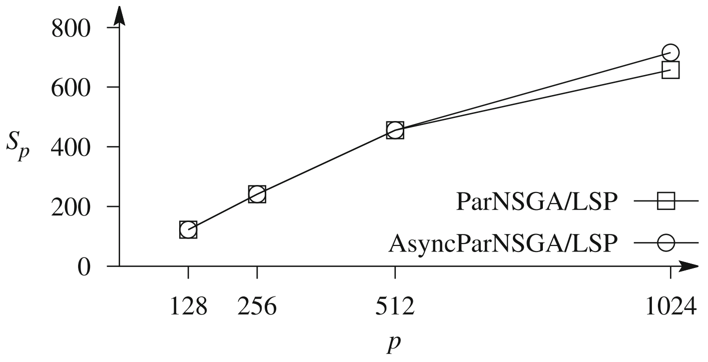


Fig. 3. Speed-up of synchronous and asynchronous versions of ParNSGA/LSP

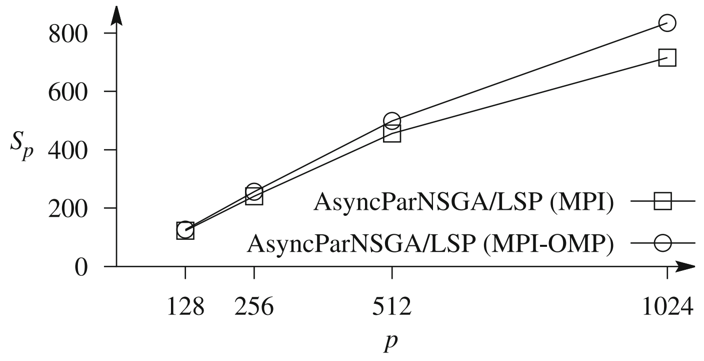


Fig. 4. Speed-up of AsyncParNSGA/LSP using distributed and hybrid distributed-shared memory parallel programming models

ory parallel computing model has significant impact on the performance of the algorithm – the speed-up using 64 groups of 16 shared memory processors has been increased by 17 %.

6 Conclusions

A hybrid (memetic) genetic algorithm for global multi-objective optimization with the local search has been parallelized, and applied to solve competitive facility location problem using a high performance computing system. Two versions of the parallel algorithm have been proposed and experimentally investigated using distributed and hybrid shared-distributed memory parallel programing models.

Results of the experimental investigation showed that, despite the additional communication costs caused by the local search, the speed-up of the hybrid algorithm is notably better than the speed-up of the parallel version of the classical NSGA-II. Results also show that utilization of the hybrid distributed-shared memory parallel programing model can significantly increase the speed-up of the parallel hybrid algorithm for global multi-objective optimization.

Acknowledgments. This research was funded by a Grant (No. MIP-063/2012) from the Research Council of Lithuania.

References

1. Friesz, T.L., Miller, T., Tobin, R.L.: Competitive networks facility location models: a survey. Pap. Reg. Sci. **65**, 47–57 (1998)
2. Plastria, F.: Static competitive facility location: an overview of optimisation approaches. Eur. J. Oper. Res. **129**(3), 461–470 (2001)
3. ReVelle, C.S., Eiselt, H.A., Daskin, M.S.: A bibliography for some fundamental problem categories in discrete location science. Eur. J. Oper. Res. **184**(3), 817–848 (2008)
4. Huff, D.L.: Defining and estimating a trade area. J. Mark. **28**, 34–38 (1964)
5. Coello, C.A.C., Lamont, G.B., Veldhuizen, D.A.V.: Evolutionary Algorithms for Solving Multi-Objective Problems, 2nd edn. Springer, New York (2007)
6. Schaffer, J.D., Grefenstette, J.J.: Multi-objective learning via genetic algorithms. In: Proceedings of the 9th International Joint Conference on Artificial Intelligence, IJCAI'85, vol. 1, pp. 593–595. Morgan Kaufmann, San Francisco (1985)
7. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans. Evol. Comput. **3**(4), 257–271 (1999)
8. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In: Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems, pp. 95–100 (2001)
9. Knowles, J.D., Corne, D.W.: Approximating the nondominated front using the Pareto archived evolution strategy. Evol. Comput. **8**(2), 149–172 (2000)
10. Srinivas, N., Deb, K.: Multiobjective optimization using nondominated sorting in genetic algorithms. Evol. Comput. **2**, 221–248 (1994)
11. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**, 182–197 (2002)
12. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P 826, California Institute of Technology (1989)
13. Mitra, K., Deb, K., Gupta, S.K.: Multiobjective dynamic optimization of an industrial nylon 6 semibatch reactor using genetic algorithms. J. Appl. Polym. Sci. **69**(1), 69–87 (1998)
14. Weile, D.S., Michielssen, E., Goldberg, D.E.: Genetic algorithm design of Pareto optimal broadband microwave absorbers. IEEE Trans. Electromagn. Compat. **38**(3), 518–525 (1996)
15. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Deb, K., Rudolph, G., Lutton, E., Merelo, J., Schoenauer, M., Schwefel, H.-P., Yao, X. (eds.) PPSN VI. LNCS, vol. 1917, pp. 849–858. Springer, Heidelberg (2000)

16. Lančinskas, A., Žilinskas, J., Ortigosa, P.M.: Local optimization in global multi-objective optimization algorithms. In: 2011 Third World Congress on Nature and Biologically Inspired Computing (NaBIC), pp. 323–328. doi:10.1109/NaBIC.2011.6089613 (2011)
17. Lančinskas, A., Ortigosa, P.M., Žilinskas, J.: Multi-objective single agent stochastic search in non-dominated sorting genetic algorithm. Nonlinear Anal. Model. Control **18**(3), 293–313 (2013)
18. Branke, J., Schmeck, H., Deb, K., Reddy, S.M.: Parallelizing multi-objective evolutionary algorithms: cone separation. In: CEC2004 Congress on Evolutionary Computation, vol. 2, pp. 1952–1957 (2004)
19. Deb, K., Zope, P., Jain, S.: Distributed computing of Pareto-optimal solutions with evolutionary algorithms. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 534–549. Springer, Heidelberg (2003)
20. Streichert, F., Ulmer, H., Zell, A.: Parallelization of multi-objective evolutionary algorithms using clustering algorithms. In: Coello Coello, C.A., Hernández Aguirre, A., Zitzler, E. (eds.) EMO 2005. LNCS, vol. 3410, pp. 92–107. Springer, Heidelberg (2005)
21. Lančinskas, A., Žilinskas, J.: Approaches to parallelize Pareto ranking in NSGA-II algorithm. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011, Part II. LNCS, vol. 7204, pp. 371–380. Springer, Heidelberg (2012)
22. Lančinskas, A., Žilinskas, J.: Solution of multi-objective competitive facility location problems using parallel NSGA-II on large scale computing systems. In: Manninen, P., Öster, P. (eds.) PARA 2012. LNCS, vol. 7782, pp. 422–433. Springer, Heidelberg (2013)

Parallelization of Encryption Algorithm Based on Chaos System and Neural Networks

Dariusz Burak(✉)

Faculty of Computer Science and Information Technology, West Pomeranian University of Technology, ul.Żołnierska 49, 71-210 Szczecin, Poland
dburak@wi.zut.edu.pl

Abstract. In this paper, the results of parallelizing of encryption algorithm based on a chaos system and neural networks are presented. A data dependence analysis of loops was applied in order to parallelize the algorithm. The parallelism of the algorithm is demonstrated in accordance with the OpenMP standard. As a result of my study, it was stated that the most time-consuming loops of the algorithm are suitable for parallelization. The efficiency measurement of a parallel program is showed.

Keywords: Neural networks · Chaotic system · Encryption algorithm · Parallelization · OpenMP

1 Introduction

One of the most important functional features of cryptographic algorithms used in industry and engineering is cipher speed. This feature is extremely important, in case of stream ciphers and block ciphers since they usually work on large data sets. Thus even not much differences of speed may cause the choice of the faster cipher by the user. Therefore, it is all-important to parallelize encryption algorithms in order to achieve faster processing using multicore processors and multiprocessing systems. In recent years, besides classical ciphers such as Rijndael, Camellia, IDEA or A5, alternative approaches of constructing ciphers based on application of the theory of chaotic dynamical systems has been developed. Futhermore neural networks are introduced to design encryption algorithms considering the complicated and time-varying nature of the structures. Chaotic neural networks are particulary suitable for data protection. Nowadays, there are many descriptions of various ciphers based on chaotic maps, for instance [1–10]. The critical issue in chaotic ciphers is program implementation.

Unlike parallel implementations of classical block ciphers, for instance AES [11], IDEA [12], there are only a few parallel implementations of chaotic block ciphers, for instance [13]. Being seemingly a research gap it is absolutely fundamental to show real functional advantages and disadvantages of the encryption algorithm using software or hardware implementation.

The main contribution of the study is developing a parallel algorithm in accordance with OpenMP of the cipher designed by Lian and Chen presented in

[14] (called further LC encryption algorithm). [14] is based on transformations of a source code written in the C language representing the sequential algorithm.

This paper is organized as follows. The next section briefly describes the LC encryption algorithm. In Sect. 3, parallelization process is fully described. In Sect. 4, the experimental results obtained for developed parallel algorithm are presented. Finally, concluding remarks are given in Sect. 5.

2 Description of the LC Encryption Algorithm

The LC encryption algorithm was published by Lian and Chen in 2011 [14]. The neural networks composed of two layers are used to construct the encryption algorithm, as shown in Fig. 1. The first layer generates the random sequences with chaos, and the second layer encrypts the data content.

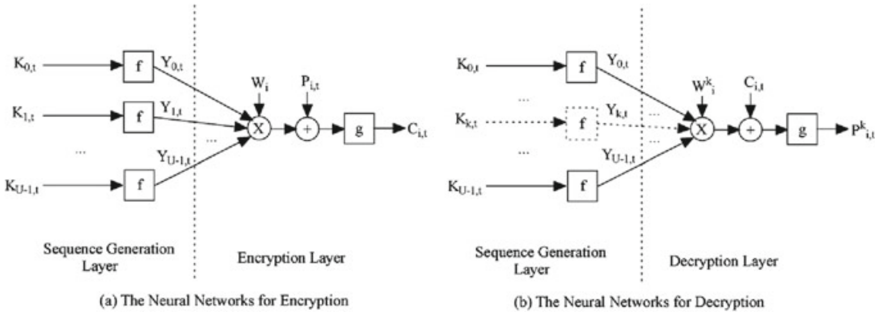


Fig. 1. The LC encryption algorithm.

According to the neural networks, the encryption operation is composed of two steps.

Firstly, the random sequences are generated as follows:
$Y_{j,t} = (f(\frac{K_{j,t}}{2^J}) - 0.5)\frac{Q}{Z}$,
$(j = 0, 1, ..., U - 1, t = 0, 1, ..., n - 1)$,
where:
Q is a integer satisfying $Q > 0$,
Z is a integer satisfying $Z > 0$,
J is a positive integer,
$f()$ is the function defined by:
$\begin{cases} f(x) = h^u(x) \\ h(x) = 4x(1-x) \end{cases}$,
where:
$h()$ is a Logistic map [15],
$h^u()$ means to iterate $h()$ for u times.
The encryption sub-key is updated in the following:
$K_{j,t+1} = K_{j,t} \oplus C_{i,t}$,

$(i = 0, 1, ..., I - 1)$,
where:
$\oplus$ is XOR operation.

Secondly, the random sequences are used to encrypt the data content. Before encryption, the plain-content is preprocessed as follows:

$$P_(i,t) = \begin{cases} \frac{Q}{2}, P_{i,t} < \frac{Q}{2}, \\ P_{i,t}, \frac{Q}{2} \leq P_{i,t} < L - \frac{Q}{2}, \\ L - \frac{Q}{2} - 1, P_{i,t} \geq L - \frac{Q}{2}, \end{cases} .$$

Then, the data content is encrypted by:
$C_{i,t} = (\sum_{j=0}^{U-1} w_{i,j} Y_{j,t} + P_{i,t}) \mod L$,
where:
$w_{i,j} = -1$ or 1 ($i = 0, 1, ..., I - 1$, and $j = 0, 1, ..., U - 1$).

The decryption process is also composed of two steps. Firstly, the random sequences are generated as follows:

$$\begin{cases} Y_{j,t}^k = f(K_{j,t}) = Y_{j,t}, s_j^k = 1 \\ Y_{j,t}^k = 0, s_j^k = 0 \end{cases} ,$$

where:
$(j = 0, 1, ..., U - 1)$.

Only the $U - Z$ sequences are generated in this way, while the other Z sequences are kept as zeros.

Secondly, the random sequences are used to decrypt the data content by:
$P_{i,t}^k = (\sum_{j=0}^{U-1} w_{i,j}^k Y_{j,t}^k + C_{i,t}) \mod L$,
where:
$w_{i,j}^k = -w_{i,j}$ $(j = 0, 1, ..., U - 1)$.

More detailed description of LC encryption algorithm is given in [14].

3 Parallelization Process of the LC Encryption Algorithm

Given the fact that proposed algorithm can work in block manner it is necessary to prepare a C source code representing the sequential LC encryption algorithm working in Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB) and Counter (CTR) modes of operation before I start parallelizing process. The source code of the LC encryption algorithm in the essential ECB mode contains sixteen *for* loops. Fourteen of them include no I/O functions. Some of these loops are time-consuming. Thus their parallelization is critical for reducing the total time of the parallel algorithm execution.

In order to find dependences in program a research tool for analyzing array data dependences called Petit was applied. Petit was developed at the University of Maryland under the Omega Project and is freely available for both DOS and UNIX systems [16,17].

The OpenMP standard was used to present parallelized loops. The OpenMP Application Program Interface (API) [18–20] supports multiplatform shared memory parallel programming in C/C++ and Fortran on all architectures including Unix and Windows NT platforms. OpenMP is a collection of compiler directives, library routines and environment variables which could be used to specify shared memory parallelism. OpenMP directives extend a sequential programming language with Single Program Multiple Data (SPMD) constructs, work-sharing constructs, synchronization constructs and help us to operate on both shared data and private data. An OpenMP program begins execution as a single task (called a master thread). When a parallel construct is encountered, the master thread creates a team of threads. The statements within the parallel construct are executed in parallel by each thread in a team. At the end of the parallel construct, the threads of the team are synchronized. Then only the master thread continues execution until the next parallel construct will be encountered. To build a valid parallel code, it is necessary to preserve all dependences, data conflicts and requirements regarding parallelism of a program [18–20].

The process of the LC encryption algorithm parallelization can be divided into the following stages:

– carrying out the dependence analysis of a sequential source code in order to detect parallelizable loops,
– selecting parallelization methods based on source code transformations,
– constructing parallel forms of program loops in accordance with the OpenMP standard.

There are the following basic types of the data dependences that occur in *for* loops: a Data Flow Dependence, a Data Anti-dependence and an Output Dependence [21–24]. Additionally, control dependence determines the ordering of an instruction i, with respect to a branch instruction so the instruction i is executed in a correct program order.

At the beginning of the parallelization process, I carried out experiments with sequential LC algorithm for a 5 megabytes input file in order to find the most time-consuming loops of this algorithm. It appeared that the algorithm has two computational bottlenecks: the first is enclosed in the function lc_enc() and the second is enclosed in the function lc_dec(). The lc_enc() function enables enciphering of the whichever number of data blocks and the lc_dec() one does the same for deciphering process (analogically to similar functions of the classic cryptographic algorithms like DES- the des_enc(), the des_dec(), LOKI91- the loki_enc(), the loki_dec or IDEA- the idea_enc(), the idea_dec() presented in [25]). Thus the parallelization of *for* loops included in these functions has a unique meaning.

The bodies of the most time-consuming loops included in these functions are the following:

Loop 3.1 of the encryption process

```
for (i=0;i<num/BUFFSIZE;i++) {
    fread(buff,sizeof(byte),BUFFSIZE,fplain);
    for (j=0;j<BUFFBLOCKS;j++) {
        for (k=0;k<U_VALUE;k++) {
            Y[k]=((logisticequation((key[k]/pow(2,J_VALUE)),
               ITERS)) - 0.5)*(Q_VALUE/Z_VALUE);
        }
        for (k=0;k<I_VALUE;k++) {
            Ywyn=0;
            for (l=0;l<U_VALUE;l++) {
                Ywyn += wages[k][l]*Y[l];
            }
            buffout[I_VALUE*j+k]=((byte)round(Ywyn*L_VALUE)+
                            (buff[I_VALUE*j+k]) )%L_VALUE;
        }
    }
    fwrite(buffout,sizeof(byte),BUFFSIZE,fcipher);
    fflush(fcipher);
}.
```

Loop 3.2 of the decryption process

```
for (i=0;i<num/BUFFSIZE;i++) {
    fread(buff,sizeof(byte),BUFFSIZE,fcipher);
    for (j=0;j<BUFFBLOCKS;j++) {
        for (k=0;k<U_VALUE;k++) {
            Y[k]=((logisticequation((key[k]/pow(2,J_VALUE)),
                ITERS)) - 0.5)*(Q_VALUE/Z_VALUE);
        }
        for (k=0;k<I_VALUE;k++) {
            Ywyn=0;
            for (l=0;l<U_VALUE;l++) {
                Ywyn += (-1)*wages[k][l]*Y[l];
            }
            buffout[I_VALUE*j+k]=((byte)round(Ywyn*L_VALUE) +
                                (buff[I_VALUE*j+k]) )%L_VALUE;
        }
    }
    write(buffout,sizeof(byte),BUFFSIZE,fplain);
    fflush(fplain);
}.
```

The declaration of constants is as follows:

```
#define U_VALUE 15 //key length (in bytes)
#define Q_VALUE 10 //parameter Q (LC encryption algorithm)
#define Z_VALUE 10 //parameter Z (LC encryption algorithm)
#define L_VALUE 256 //corresponding to 256 ASCII characters
#define I_VALUE 12 //block size (in bytes)
#define J_VALUE 8 //parameter J (LC encryption algorithm)
```

```
#define ITERS 31 //the number of iterations of the Logistic map
#define BUFFBLOCKS 5 //number of blocks allocated in data buffer
#define BUFFSIZE (BUFFBLOCKS*I_VALUE).
```

The declaration of variables is the following:

```
int i, j, k, l, num, wages[I_VALUE][U_VALUE];
double Ywyn, Y[U_VALUE];
byte buff[BUFFSIZE], buffout[BUFFSIZE], key[U_VALUE];
FILE *fplain, *fcipher;.
```

The logisticequation() function definition:

```
double logisticequation(double initial,int iter) {
int i;
    for (i=0;i<iter;i++)
        initial=(double)4*initial*(1-initial);
return initial;
}.
```

Taking into account the strong similarity of loops 3.1 and 3.2, I examined only the loop 3.1. Subsequently this analysis is valid in case of the loop 3.2.

The actual parallelization process of the loop 3.1 consists of the six following stages:

- separation of the Sequence Generation Layer from Encryption Layer; all calculations placed in Sequence Generation Layer (see bellow for the first loop of parallel form of 3.1 loop) has to be executed before starting the processing for the next layer;
- removal of multiplication from Encryption Layer; multiplication has to be calculated immediately after all calculations placed in Sequence Generation Layer are completed (see bellow for the first loop of parallel form of loop 3.1);
- indexing variable privatization (k) using OpenMP (based on the results of data dependence analysis) for the first loop;
- adding appropriate OpenMP directive and clauses (#pragma omp parallel for private() shared()) for the first loop;
- suitable variables privatization (j,k,l,Ywyn) using OpenMP (based on the results of data dependence analysis) for the nested loop (indexing by j) in the case of the third loop (see bellow);
- adding appropriate OpenMP directive and clauses (#pragma omp parallel for private() shared()) for the third loop.

The steps above result in the following parallel form of 3.1 loop in accordance with the OpenMP standard:

```
//the sequence generation layer loop
#pragma omp parallel for private(k) shared(Y,key)
```

```
for (k=0;k<U_VALUE;k++) {
    Y[k]=((logisticequation((key[k]/pow(2,J_VALUE)),ITERS))
        - 0.5)*(Q_VALUE/Z_VALUE);
}

//the multiplication loop
for (k=0;k<I_VALUE;k++) {
    _Ywyn[k]=0;
    for (l=0;l<U_VALUE;l++) {
        _Ywyn[k] += wages[k][l]*Y[l];
    }
}

//the encryption layer without multiplication loop
for (i=0;i<num/BUFFSIZE;i++) {
    fread(buff,sizeof(byte),BUFFSIZE,fplain);
    #pragma omp parallel for private(j,k,l,Ywyn)
                            shared(Y,buffout,wages,buff)
    for (j=0;j<BUFFBLOCKS;j++) {
        for (k=0;k<I_VALUE;k++) {
            buffout[I_VALUE*j+k]=((byte)round(_Ywyn[k]*L_VALUE)
                                + (buff[I_VALUE*j+k]) )%L_VALUE;
        }
    }
    fwrite(buffout,sizeof(byte),BUFFSIZE,fcipher);
    fflush(fcipher);
}.
```

The loop 3.2 was parallelized in a similar way as the loop 3.1.

4 Experimental Results

In order to study the efficiency of the presented LC parallel code I used eight Quad-Core Intel Xeon Processors 7310 Series - 1.60 GHz and the Intel C++ Compiler ver. 12.1 (that supports the OpenMP 3.1). The results received for a 20 MB input file using two, four, eight, sixteen and thirty-two cores versus the only one have been shown in Table 1. The number of threads is equal to the number of processors.

The total running time of the LC algorithm consists of the following operations:

- data receiving from an input file,
- sub-keys generation,
- data encryption,
- data decryption,
- data writing to an output file (both encrypted and decrypted text).

Table 1. Speed-up of the parallel LC algorithm in the ECB mode of operation.

Number of threads	Speed-up of the encryption process	Speed-up of the decryption process	Speed-up of the whole LC algorithm
1	1.00	1.00	1.00
2	1.96	1.99	1.41
4	3.80	3.90	1.92
8	6.20	6.30	2.34
16	6.40	6.40	2.48
32	6.20	6.20	2.30

Thus the total speed-up of the LC parallel algorithm depends heavily on the five factors:

- the degree of parallelization of the loop included in the lc_enc() function (3.1),
- the degree of parallelization of the loop included in the lc_dec() function (3.2),
- the method of reading data from an input file,
- the method of writing data to an output file,
- the block size of the LC encryption algorithm.

The results confirm that the loops included both the lc_enc() and the lc_dec() functions are parallelizable with high speed-up (see Table 1).

The block method of reading data from an input file and writing data to an output file was used. The following C language functions and block sizes was applied: fread() function and 4096-bytes block for data reading and fwrite() function and 256-bytes block for data writing.

Table 2. Speed-ups of the parallel LC algorithms in the CTR, CBC and CFB mode of operation.

Number of threads	Operation	Speed-up of the CTR mode of operation	Speed-up of the CBC mode of operation	Speed-up of the CFB mode of operation
1	Encryption	1.00	1.00	1.00
1	Decryption	1.00	1.00	1.00
2	Encryption	1.90	1.00	1.00
2	Decryption	1.90	1.90	1.90
4	Encryption	3.50	1.00	1.00
4	Decryption	3.37	3.70	3.70
8	Encryption	6.00	1.00	1.00
8	Decryption	6.10	6.10	6.10
16	Encryption	6.20	1.00	1.00
16	Decryption	6.30	6.20	6.20
32	Encryption	6.00	1.00	1.00
32	Decryption	6.10	6.00	6.00

Using the fwrite() function is crucial; choosing, for example, the fprintf() function I got much longer time of executing tasks.

During experiments I chose the block size equal to 32 bytes. My tests showed that this size of block gives a good encryption/decryption speed of the LC encryption algorithm.

In accordance with Amdahl's Law [26] the maximum speed-up of the LC encryption algorithm is limited to 4.524, because the fraction of the code that cannot be parallelized is 0.221.

I also parallelized the LC encryption algorithm in the CTR, CBC and CFB modes of operation (based on recommendation detailed described in [27]) (when possible). The results are presented in Table 2.

When the LC algorithm operates in the ECB and CTR modes of operation, both the encryption and decryption processes are parallelizable and speed ups of the whole algorithm are similar (see- Table 2). For the CBC and CFB modes only the decryption process is parallelized so the values of speed-up are lower than for the ECB and CTR modes of operation (see- Table 2).

5 Conclusions

In this paper, I describe the parallelization process of the LC encryption algorithm which was divided into parallelizable and unparallelizable parts. I have shown that the time-consuming *for* loops included in the functions responsible for the encryption and decryption processes are parallelizable. Altogether I parallelized eleven loops (from among sixteen ones). The experiments have shown that the application of the parallel LC encryption algorithm for multiprocessor and multi-core computers would considerably boost the time of the data encryption and decryption. I believe that the speed-ups received for these operations are satisfactory. Moreover, the developed parallel LC encryption algorithm can be also helpful for hardware implementations

References

1. Habutsu, T., Nishio, Y., Sasase, I., Mori, S.: A secret key cryptosystem using a chaotic map. Trans. IEICE Jpn. **E73**(7), 1041–1044 (1990)
2. Fridrich, J.: Symmetric ciphers based on two-dimensional chaotic maps. Int. J. Bifurcat. Chaos **8**(6), 1259–1284 (1998)
3. Scharinger, J.: Fast encryption of image data using chaotic Kolmogorov flows. J. Electron. Imaging **7**(2), 318–325 (1998)
4. Kocarev, L., Jakimoski, G.: Logistic map as a block encryption algorithm. Phys. Lett. A **289**(4–5), 199–206 (2001)
5. Yi, X., Tan, C.H., Siew, C.K.: A new block cipher based on chaotic tent maps. EEE Trans. Circuits Syst. I: Fundam. Theory Appl. **49**(12), 1826–1829 (2002)
6. Chen, G., Mao, Y.B., Chui, C.K.: A symmetric image encryption scheme based on 3D chaotic cat maps. Chaos Solitons Fractals **12**, 749–761 (2004)
7. Mao, Y.B., Chen, G., Lian, S.G.: A novel fast image encryption scheme based on the 3D chaotic baker map. Int. J. Bifurcat. Chaos **14**(10), 3613–3624 (2004)

8. Lian, S., Sun, J., Wang, Z.: A block cipher based on a suitable use of the chaotic standard map. Chaos, Solitons and Fractals **26**(1), 117–129 (2005)
9. Xua, S., Wang, J., Yang, S.: A novel block cipher based on chaotic maps. In: Congress on Image and Signal Processing, vol. 3 (2008)
10. Pareek, N.K., Patidar, V., Sud, K.K.: Block cipher using 1D and 2D chaotic maps. Int. J. Inf. Commun. Technol. **2**(3) (2010)
11. Bielecki, W.: Exploiting loop-level parallelism in the AES algorithm. WSEAS Trans. Comput. **5**(1), 125–133 (2006)
12. Beletskyy, V., Burak, D.: Parallelization of the IDEA algorithm. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2004. LNCS, vol. 3036, pp. 635–638. Springer, Heidelberg (2004)
13. Burak, D., Chudzik, M.: Parallelization of the discrete chaotic block encryption algorithm. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011, Part II. LNCS, vol. 7204, pp. 323–332. Springer, Heidelberg (2012)
14. Lian, S., Chen, X.: Traceable content protection based on chaos and neural networks. Appl. Soft Comput. **11**(7), 4293–4301 (2011)
15. Zhusubaliyev, Z.T., Mosekilde, E.: Bifurcations and Chaos in Piecewise-smooth Dynamical Systems. World Scientific Publishing Co., Pte. Ltd., Singapore (2003)
16. Kelly, W., Maslov, V., Pugh, W., Rosser, E., Shpeisman, T., Wonnacott, D.: New User Interface for Petit and Other Extensions. User Guide (1996)
17. The Omega Project: Frameworks and Algorithms for the Analysis and Transformation of Scientific Programs. http://www.cs.umd.edu/projects/omega/
18. Chapman, B., Jost, G., van der Pas, R.: Using OpenMP - Portable Shared Memory Parallel Programming. The MIT Press, Cambridge (2007)
19. Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., Menon, R.: Parallel Programming in OpenMP. Morgan Kaufmann Publishers, Inc., San Francisco (2001)
20. OpenMP Application Program Interface. Version 3.1 July 2011 (2011)
21. Moldovan, D.I.: Parallel Processing: From Applications to Systems. Morgan Kaufmann Publishers, Inc., San Mateo (1993)
22. Muchnick, S.S.: Advanced Compiler Design and Implementation. Morgan Kaufmann Publishers Inc., San Francisco (1997)
23. Allen, R., Kennedy, K.: Optimizing Compilers for Modern Architectures: A Dependencebased Approach. Morgan Kaufmann Publishers, Inc., San Francisco (2001)
24. Aho, A., Lam, M., Sethi, R., Ullman, J.: Compilers: Principles, Techniques, and Tools, 2nd edn. Prentice Hall, Upper Saddle River (2006)
25. Schneier, B.: Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edn. Wiley, New (1995)
26. Amdahl, G.M.: Validity of the single-processor approach to achieving large scale computing capabilities. In: AFIPS Conference Proceedings, pp. 483–485 (1967)
27. Dworkin, M.: Recommendation for block cipher modes of operation: methods and techniques. NIST Special, Publication 800–38A, Dec (2001)

Minisymposium on HPC Applications in Physical Sciences

Simulations of the Adsorption Behavior of Dendrimers

Jarosław S. Kłos[1,3](✉) and Jens U. Sommer[1,2]

[1] Leibniz Institute of Polymer Research, Dresden e. V., 01069 Dresden, Germany
klos@ipfdd.de

[2] Institute for Theoretical Physics, Technische Universität Dresden, 01069 Dresden, Germany

[3] Faculty of Physics, Adam Mickiewicz University, Umultowska 85, 61-614 Poznań, Poland

Abstract. Using Monte Carlo simulations we study adsorption of dendrimers with flexible spacers onto a flat surface in a wide range of molecular weight, N, generation number, G, spacer length, S, and the monomer-surface interaction strength parameter, τ. Our calculations indicate that for large values of N the dendrimers exist in three τ-dependent regions referred to as non-adsorbed, critical and adsorbed. Slightly below the critical point of adsorption, τ_c, a weakly adsorbed state is approached in which the molecules stick to the surface and are spherical in shape. By further lowering τ below a spacer-length dependent value, $\tau^*(S) < \tau_c$, a jumplike transition into a strongly adsorbed state occurs. Here, the dendrimers become flat and their lateral size is described by a 2D mean-field model.

Keywords: Polymer · Dendrimer · Adsorption · Simulation

1 Introduction

For the sake of design of nanodevices with optimal transport properties it is crucial to understand the behavior of dendrimers near surfaces. Adsorption of PAMAM (polyamidoamine) dendrimers was examined with the use of atomic force microscopy and optical reflectometry [1,2]. Experiments also considered the influence of ionic strength and solution pH on the molecular dimension of PAMAMs and their structural properties at the interface between an aqueous solution and a hydrophobic or hydrophilic substrate [3–5].

Lattice simulations showed that adorbed dendrimers spread out and flatten down on the surface with increasing interaction strength. It was reported that depending on G and the interaction strength the molecules exist in five different configurational states: A desorption region, a weak adsorption region in which the shape of adsorbed dendrimers is only weakly perturbed and three more regions in which the dendrimers display different conformational features [6]. Using Brownian dynamics simulations the effects of the strength of hydrophobicity, dendrimer generation and hydrophobicity distributions on the dendrimer

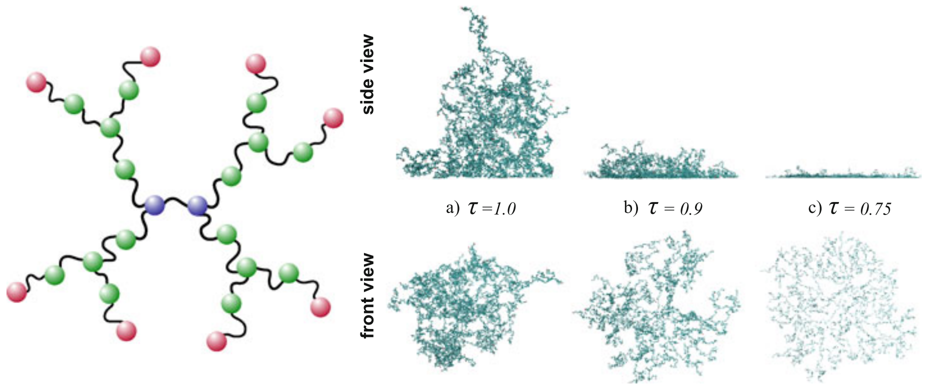


Fig. 1. LEFT: Dendrimer architecture for $G = 2$ and $S = 2$. RIGHT: Snapshots of the $G6$ dendrimer with spacer $S = 32$ at (a) $\tau = 1.0$, (b) $\tau = 0.9$, (c) $\tau = 0.75$. Red (blue) circles represent the terminal groups (the core) (Color figure online).

conformation, monomer distributions within dendrimers and near the surface were examined as well [7]. Brownian dynamics simulations were carried out to inspect adsorption of charged dendrimers onto oppositely charged flat surfaces for various screening lengths, dendrimer generation, and dendrimer charge distribution [8].

In this work we consider adsorption of dendrimers using Monte Carlo simulations based on the bond fluctuation model (BFM) [9–12]. While the adsorption behavior of linear chains is well understood in terms of surface critical phenomena [13], the influence of dendrimer architecture, see Fig. 1 LEFT, on adsorption is still an open problem.

2 Model and Simulation Details

In the BFM we used, a bead is a simple cube represented by eight lattice sites on a simple cubic lattice. Beads are connected with bonds so as to form the required molecular architecture. In the model different beads must not occupy even one same lattice site due to the excluded volume condition, and the bond vectors have to belong to a certain set of allowed vectors only. The allowed vectors are chosen is such a way that the excluded volume condition prevents them from crossing each other. In the framework of the Monte Carlo approach trial moves are performed by moving a bead by one (out of six) lattice site and by testing the new conformation and bonds against the above mentioned restrictions on the beads and bonds [9]. Since we studied the adsorption behavior of single dendrimers the calculations were carried out using the serial version of the BFM only. It should be stressed that two parallizable versions of the BFM were already implemented on GPU hardware. The codes were tested for various static and dynamic properties of dense polymer system and compared with the standard implementation. The parallel implementations of the BFM on graphics processors were reported to outperform by a factor of up to 50 times an equivalent implementation on single CPU processor [14].

We examined single $G3$–$G7$ dendrimers with the core of two bonded units, branching functionality $f = 3$ and spacer length $S = 1, 2, 4, 8, 16, 32$ in a cubic box with periodic boundaries in the x- and y- directions and with an impenetrable wall at $z = 0$. One of the dendrimers' terminal groups was immobilized and attached at the adsorbing surface at $z = 0$. Trial conformations were generated and accepted using the bond fluctuation model along with the standard Metropolis test [15]. Each bead was subject to an attraction exerted on it by the surface. Whenever a bead touched (detouched from) the surface the system energy decreased (increased) by δ, which leads to the reduced temperature

$$\tau = \frac{k_B T}{\delta} \quad , \tag{1}$$

where k_B and T denote the Boltzmann constant and the absolute temperature. Moves at (off) the surface were always accepted. Depending on G and S the dendrimers were equilibrated for a maximum of 10^7 MCS (Monte Carlo Steps; in one MCS on average each monomer is selected to be moved in a randomly chosen, one of the six directions by a single lattice unit), whereas averages were calculated for 10^3–10^5 equilibrium configurations stored every 10^4th MCS. An equilibrium state was considered achieved once the means of various measured quantities characterizing the molecules such as the radius of gyration and the number of adsorbed monomers did not reveal systematic changes. Throughout the paper the errors we calculated using the rebinning method [16] are smaller than the smbols' size. By carrying out two independent runs based on different sequences of random numbers and initiated with two different starting conformations of the $G5$ dendrimer with spacers $S = 8$ and $S = 32$ at three different temperatures, we also successfully checked the method for its convergence. The results from both calculations agreed with each other within statistical fluctuations. Even at the highest adsorption energy considered monomer movements did not reveal any freezing effect (note that adsorbed monomers are always movable in the direction parallel to the substrate). To avoid kinetic trapping gradual lowering of τ was applied.

3 Results

3.1 Adsorption Transition

To examine the adsorption behavior of dendrimers we first consider the adsorption order parameter $m = M/N$, where M denotes the number of monomers in contact with the surface [13]. In Fig. 2 we display this quantity as a function of τ at the fixed generation number G. It is clearly seen that the dendrimers display an adsorption transition manifested through an increase in m from nearly zero in the extreme of high τ up to some finite values approaching one at low τ-values. Obviously, in the limit of low τ adsorption is complete and all monomers are in contact with the surface. Based on the order parameter we can roughly

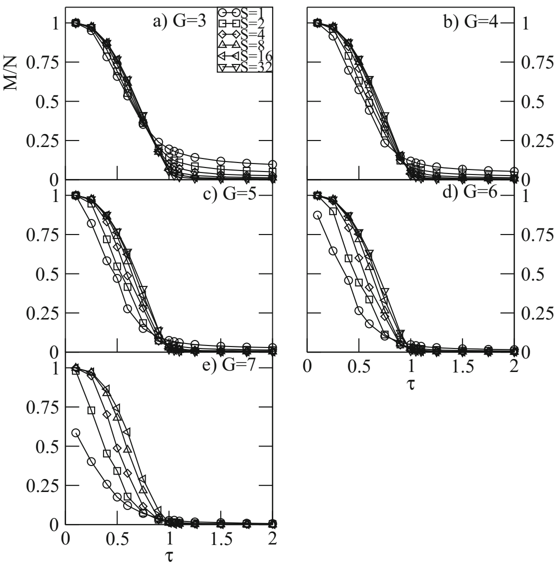


Fig. 2. Fraction of adsorbed monomers M/N vs τ at fixed G and varying S.

distinguish between the three regions: The crossover from the non-adsorbed to the adsorbed state at around $\tau \simeq 1$, see snapshot in Fig. 1(a) RIGHT. For lower values of τ the number of adsorbed monomers increases sharply with lowering τ, see Fig. 1(b) RIGHT. For very low τ almost all of the monomers are on the surface ($m \simeq 1$), see Fig. 1(c) RIGHT. Note that due to packing constraints in 2D for the $G7$-dendrimer with the shortest spacer, $S = 1$, the order parameter saturates at $m \approx 0.6$.

By analogy with adsorption of linear chains we consider m as a function of N at constant τ [13,17]. Under the assumption of criticality and for large values of N the behavior of $m(N)$ splits into three classes according to three regimes (non-adsorbed, critical, adsorbed):

$$m \sim \begin{cases} N^{-1} & \text{for } \tau > \tau_c \\ N^{\phi-1} & \text{for } \tau = \tau_c \\ N^{0} & \text{for } \tau < \tau_c. \end{cases} \tag{2}$$

In Fig. 3 LEFT we plot $m(N)$ for fixed G under the variation of S. The splitting of isotherms into the three asymptotic classes is clearly visible. As the best estimate for the critical point we can take $1.0 \leq \tau_c \leq 1.05$. It is interesting to note that for $G > 4$, see Fig. 3(c)–(e) LEFT, the isotherms for $\tau = 0.9$ change from non-adsorption to adsorption behavior with increasing S. Increasing S reduces excluded volume constraints and increases the strength of adsorption of individual spacers which drives the dendrimer from the non-adsorbed into an adsorbed asymptotics. In Fig. 3 RIGHT we plot $m(N)$ for fixed S with increasing G. For $S = 1$ the asymptotic behavior tends to non-adsorbed independently

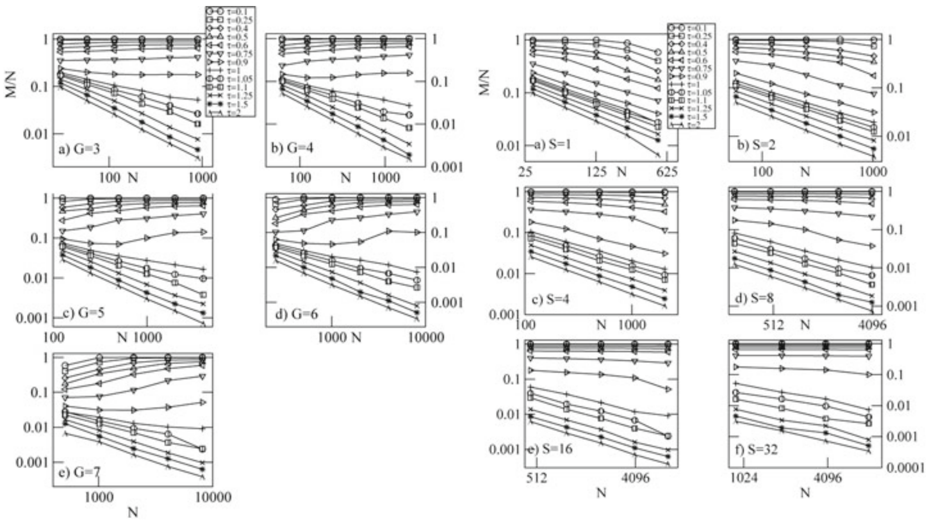


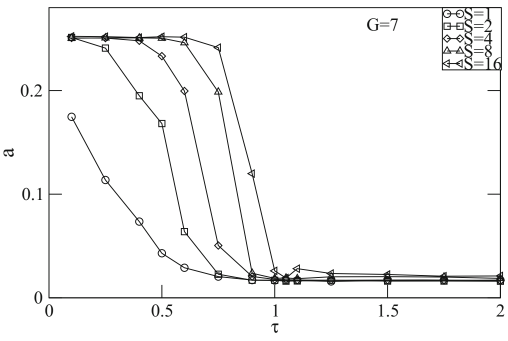
Fig. 3. LEFT: (RIGHT:) Isotherms of the order parameter M/N vs N at fixed G (S).

of τ. This illustrates the strong excluded volume interactions which results in packing-dominated conformations for large G. Also, for large S-values m tends to non-adsorbed with sufficiently high G as a consequence of packing effects.

3.2 Shape and Size of Dendrimers

In the strongly adsorbed state dendrimers are not isotropic. In order to analyze the shape of the dendrimers we consider the relative shape anisotropy, a. In particular, $a = 0$, $a = 1/4$ and $a = 1$ for spherical, oblate and extremely elongated ellipsoides, respectively [18–20]. In Fig. 4 LEFT we display $a\,(\tau)$ for various spacers at $G = 7$. A jumplike transition scenario corresponding to a collapse of the structure (from nearly spherical to flat) in the direction perpendicular to the surface is seen for long spacers. For $G = 7$ one can observe the shift in the transition with respect to S. For $S = 16$ the collapse is centred at about $\tau^*(S = 16) = 0.9$ which is reduced to $\tau^*(S = 8) = 0.75$ for $S = 8$. For low τ-values the shape anisotropy reaches $a \approx 0.25$ which signals that the polymers are oblate. Furthermore, we note that $\tau^* < \tau_c$, i.e., the shape-transition temperatures are below the critical point of adsorption. In particular, slightly below τ_c the dendrimer is in the weakly adsorbed state where its shape remains spherical.

Since the adsorbing interface breaks the isotropy of the system, we have to distinguish between the parallel, $R_{g\|}$, and perpendicular, $R_{g\perp}$, components of the dendrimers' radius of gyration, R_g. At high τ the dendrimers are nearly isotropic and their extension corresponds to the 3D behavior. In our previous work we showed [12] that the mean field argument [21] is able to describe the



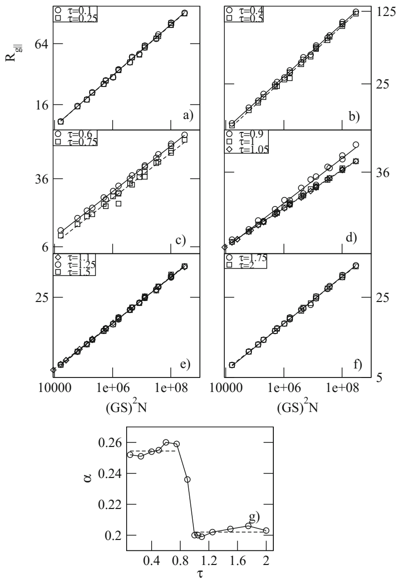


Fig. 4. LEFT: $a\,(\tau)$ for various spacers and $G = 7$. RIGHT: (a–f) Scaling behavior of $R_{g\|}$ at the considered τ-values for $S \geq 4$. The lines are fits to the data points of the form Eq. 6. (g) Exponent α vs τ. The horizontal dashed lines indicate α obtained from averaging the results in the respective asymptotic regions (2D and 3D).

scaling of the free dendrimer's size in a good solvent according to

$$R_{g\perp} \sim R_{g\|} \sim \left[(GS)^2 N\right]^{1/5} \quad for \quad \tau \gg \tau_c. \tag{3}$$

On the other hand in the limit of low τ-values when the dendrimers are nearly 2D objects we expect the free energy to take the form (with unit-carrying pre-factors omitted, length units are taken in units of a Kuhn segmet)

$$\frac{F}{k_B T} = \frac{R_{g\|}^2}{GS} + \frac{GSN}{R_{g\|}^2} \quad , \tag{4}$$

leading to

$$R_{g\|} \sim \left[(GS)^2 N\right]^{1/4} \quad for \quad \tau \ll \tau_c. \tag{5}$$

In Fig. 4(a)–(f) we display $R_{g\|}$ as a function of $(GS)^2 N$ at various τ. According to Eqs. 3 and 5 we fit the data to a power law

$$R_{g\|} \sim \left[(GS)^2 N\right]^{\alpha} \quad . \tag{6}$$

It is seen from Fig. 4(a)–(f) that the proposed scaling relation is well obeyed for all G and $S \geq 4$, except for the intermediate region $0.75 \leq \tau \leq 0.9$ where some discrepancies occur. In Fig. 4(g) we plot $\alpha(\tau)$. In the strong adsorption region we obtain $\alpha \approx 0.25$ with a slight tendency to increase as the intermediate region is approached. At higher τ-values around $\tau = 0.9$ the exponent decreases sharply to $\alpha \approx 0.20$ as the weak adsorption region is entered. Both limiting values correspond to the predicted asymptotic behavior in 2D and 3D and crossover between both regions is jumplike.

4 Summary and Conclusions

Using the bond fluctuation model we have investigated adsorption of dendrimers onto a flat surface. By analyzing the adsorption order parameter we have found that, depending on the reduced temperature τ, the molecules can exist in three states referred to as non-adsorbed, critical and adsorbed, respectively. Our results indicate that below the critical temperature of adsorption, $\tau_c \approx 1.01$, a number of monomers are adsorbed while the dendrimer as a whole is only weakly perturbed and retains its spherical shape. At a characteristic temperature, $\tau^* < \tau_c$, the dendrimer undergoes a jumplike transition into a flat conformation. This shape transition is demonstrated by the behavior of the shape anisotropy of the dendrimer which takes values corresponding to flat objects $a \approx 0.25$. In the flatly adsorbed state the lateral extension of the dendrimer follows the scaling behavior for a 2D-object. According to the mean-field prediction we obtain a scaling of $R_{g\|} \sim N^{1/4}$.

Acknowledgments. Support from DFG contract numbers SO-277/2-1 and KL 2470/1-1 is acknowledged. Part of the calculations were carried out at ZIH of the TU Dresden.

References

1. Li, J., Piehler, L.T., Qin, D., Baker, J.R., Tomalia, D.A.: Visualization and characterization of poly(amidoamine) dendrimers by atomic force microscopy. Langmuir **16**(13), 5613–5616 (2000)
2. Longtin, R., Maroni, P., Borkovec, M.: Transition from completely reversible to irreversible adsorption of poly(amido amine) dendrimers on silica. Langmuir **25**, 2928–2934 (2009)
3. Pericet-Camara, R., Papastavrou, G., Borkovec, M.: Atomic force microscopy study of the adsorption and electrostatic self-organization of poly(amidoamine) dendrimers on mica. Langmuir **20**(8), 3264–3270 (2004)
4. Betley, T.A., Holl, M.M.B., Orr, B.G., Swanson, D.R., Tomalia, D.A., Baker, J.R.: Tapping mode atomic force microscopy investigation of poly(amidoamine) dendrimers: effects of substrate and PH on dendrimer deformation. Langmuir **17**(9), 2768–2773 (2001)
5. Müller, T., Yablon, D.G., Karchner, R., Knapp, D., Kleinman, M.H., Fang, H., Durning, C.J., Tomalia, D.A., Turro, N.J., Flynn, G.W.: AFM studies of high-generation PAMAM dendrimers at the liquid/solid interface. Langmuir **18**(20), 7452–7455 (2002)

6. Mansfield, M.L.: Surface adsorption of model dendrimers. Polymer **37**(17), 3835–3841 (1996)
7. Suman, B., Kumar, S.: Brownian dynamics simulations of hydrophobic dendrimer adsorption. Mol. Simul. **35**(1–2), 38–49 (2009)
8. Suman, B., Kumar, S.: Adsorption of charged dendrimers: a Brownian dynamics study. J. Phys. Chem. B **111**, 8728 (2007)
9. Deutsch, H.P., Binder, K.: Interdiffusion and self-diffusion in polymer mixtures: a Monte Carlo study. J. Chem. Phys. **94**(3), 2294–2304 (1991)
10. Carmesin, I., Kremer, K.: The bond fluctuation method: a new effective algorithm for the dynamics of polymers in all spatial dimensions. Macromolecules **21**(9), 2819 (1988)
11. Trautenberg, H.L., Hölzl, T., Göritz, D.: Evidence for the absence of bond-crossing in the three-dimensional bond fluctuation model. Comput. Theor. Polym. Sci. **6**, 135 (1996)
12. Kłos, J.S., Sommer, J.U.: Properties of dendrimers with flexible spacer-chains: a Monte Carlo study. Macromolecules **42**, 4878 (2009)
13. Eisenriegler, E., Kremer, K., Binder, K.: Adsorption of polymer chains at surfaces: scaling and Monte Carlo analyses. J. Chem. Phys. **77**(12), 6296–6320 (1982)
14. Nedelcu, S., Werner, M., Lang, M., Sommer, J.U.: GPU implementations of the bond fluctuation model. J. Comp. Phys. **231**(7), 2811–2824 (2012)
15. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. J. Chem. Phys. **21**, 1087 (1953)
16. Ambegaokar, V., Troyer, M.: Estimating errors reliably in Monte Carlo simulations of the ehrenfest model. Am. J. Phys. **78**, 150–157 (2010)
17. Descas, R., Sommer, J.U., Blumen, A.: Grafted polymer chains interacting with substrates: computer simulations and scaling. Macrom. Theor. Sim. **17**, 429–453 (2008)
18. Theodorou, D.N., Suter, U.: Shape of unperturbed linear polymers: polypropylene. Macromolecules **18**, 1206–1214 (1985)
19. Rudnick, J., Gaspari, G.: The asphericity of random walks. J. Phys. A: Math. Gen. **19**, L191–L193 (1986)
20. Maiti, P.K., Çağin, T., Wang, G., Goddard III, W.: Structure of pamam dendrimers: generations 1 through 11. Macromolecules **37**, 6236–6254 (2004)
21. Boris, D., Rubinstein, M.: A self-consistent mean field model of a starburst dendrimer: dense core vs dense shell. Macromolecules **29**, 7251–7260 (1996)

An Optimized Lattice Boltzmann Code for BlueGene/Q

Marcello Pivanti[1], Filippo Mantovani[2], Sebastiano Fabio Schifano[1](✉), Raffaele Tripiccione[1], and Luca Zenesini[1]

[1] Università di Ferrara and INFN, Ferrara, Italy
schifano@fe.infn.it
[2] Facultät für Physik, Univesität Regensburg, Regensburg, Germany

Abstract. In this paper we describe an optimized implementation of a Lattice Boltzmann (LB) code on the BlueGene/Q system, the latest generation massively parallel system of the BlueGene family. We consider a state-of-art LB code, that accurately reproduces the thermo-hydrodynamics of a 2D-fluid obeying the equations of state of a perfect gas. The regular structure of LB algorithms offers several levels of algorithmic parallelism that can be matched by a massively parallel computer architecture. However the complex memory access patterns associated to our LB model make it not trivial to efficiently exploit all available parallelism. We describe our implementation strategies, based on previous experience made on clusters of many-core processors and GPUs, present results and analyze and compare performances.

Keywords: Lattice Boltzmann · High performance computing · Massively parallel architectures · Performance Analysis

1 Introduction

Computational techniques are ubiquitous in fluid-dynamics to obtain reliable solutions to the non-linear equations of motion in regimes interesting for physics or engineering. Over the years, several numerical approaches have been theoretically developed and implemented on massively parallel computers.

The Lattice Boltzmann (LB) method is a flexible approach, able to cope with many different fluid equations (e.g., multi-phase, multicomponent and thermal fluids) and with complex geometries or boundary conditions. LB builds on the fact that the details of the microscopic-scale interaction among fluid components do not change the structure of the equations of motion at the macroscopic scale, but only modulate the values of the macroscopic parameters (e.g., viscosity). LB then simulates some synthetic dynamics of fictitious particles that, appropriately averaged, provides the correct values of the macroscopic observables of the flow; see [1] for an introduction.

LB schemes are local (they do not require the computation of non local fields – e.g. pressure– and interactions in coordinate space are among nearest neighbors), so they are a natural target for efficient implementations on massively parallel systems. Starting from early attempts on experimental machines [2], recent works have focused on parallelizing the code both inside one processing node and across many nodes [3,4]. GPUs implementations have been reported in [5] and early experience with many-core processors has been studied in [6].

In this paper we discuss and apply optimization strategies for the Blue-Gene/Q (BG/Q) massively parallel system; in this machine a good match in combining inter-node and intra-node parallelism is needed to obtain satisfactory performance. We consider a state-of-the-art LB method (described below) which has a large computational load and a non trivial pattern of memory access.

Following this overview section, this paper briefly reviews the LB algorithm that we consider and then describes the main features of BG/Q. A detailed description of our LB implementation follows, while the next section presents and comments performance results. The paper ends with our concluding remarks.

2 The Lattice Boltzmann Methods

In this section we briefly describe the Lattice Boltzmann approach to the numerical solution of a class of Navier-Stokes equations that describe fluid flows. The Thermal-Kinetic description in D dimensions of a compressible gas/fluid of variable density, ρ, local velocity $\boldsymbol{u}$, internal energy, $\mathcal{K}$ and subject to a local body force density, $\boldsymbol{g}$ (gravity), is given by the following equations:

$$\partial_t \rho + \partial_i(\rho u_i) = 0 \tag{1}$$

$$\partial_t(\rho u_k) + \partial_i(P_{ik}) = \rho g_k \tag{2}$$

$$\partial_t \mathcal{K} + \frac{1}{2}\partial_i q_i = \rho g_i u_i \tag{3}$$

where P_{ik} and q_i are the momentum and energy fluxes; the indexes identify space coordinates, and sums over repeated indexes is implied.

In the continuum, one shows that it is possible to recover exactly these equations, starting from the Boltzmann equation and introducing a suitable shift of the velocity and temperature fields entering in the local equilibrium [7]. The discretized counterpart of the continuum description (adopted in this paper) uses a set of fields $f_l(x,t)$ associated to so-called *populations*; the latter can be visualized as pseudo-particles moving in appropriate directions on a discrete mesh (see Fig. 1). The master evolution equation in the discrete mesh is:

$$f_l(\boldsymbol{x} + \boldsymbol{c}_l \Delta t, t + \Delta t) - f_l(\boldsymbol{x}, t) = -\frac{\Delta t}{\tau}\left(f_l(\boldsymbol{x}, t) - f_l^{(eq)}\right); \tag{4}$$

subscript l runs over the discrete set of velocities, $\boldsymbol{c}_l$ (see again Fig. 1), and equilibrium population densities depend on the hydro-dynamical (macroscopic) fields on the lattice, $f_l^{(eq)} = f_l^{(eq)}(\boldsymbol{x}, \rho, \bar{\boldsymbol{u}}, \bar{T})$.

In this paper we consider a state-of-the-art LB method that correctly describes the behavior in two dimensions of a compressible fluid that obeys the equation of state of a perfect gas. All details are in [7,8]. This algorithm uses 37 populations (it is a so called D2Q37 model), much more than necessary in other established LB methods (e.g., D2Q9 or D3Q19).

In short, macroscopic fields are defined in terms of the LB populations: $\rho = \sum_l f_l$, $\rho \boldsymbol{u} = \sum_l \boldsymbol{c}_l f_l$, $D\rho T = \sum_l |\boldsymbol{c}_l - \boldsymbol{u}|^2 f_l$. When going into all mathematical details, one finds that shifts and renormalizations have to be applied to the averaged hydrodinamical quantities to correct for lattice discretization effects. After performing these manipulations, one recovers the correct thermo-hydrodynamical equations :

$$D_t \rho = -\rho \partial_i u_i^{(H)} \tag{5}$$

$$\rho D_t u_i^{(H)} = -\partial_i p - \rho g \delta_{i,2} + \nu \partial_{jj} u_i^{(H)} \tag{6}$$

$$\rho c_v D_t T^{(H)} + p \partial_i u_i^{(H)} = k \partial_{ii} T^{(H)} \tag{7}$$

where we introduce the material derivative, $D_t = \partial_t + u_j^{(H)} \partial_j$, and neglect viscous dissipation in the heat equation (usually small); superscript H denotes the lattice-corrected quantities, c_v is the specific heat at constant volume for an ideal gas $p = \rho T^{(H)}$, and ν and k are the transport coefficients.

LB methods have a large degree of available parallelism. Defining $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{c}_l \Delta t$ and rewriting the main evolution equation as:

$$f_l(\boldsymbol{y}, t + \Delta t) = f_l(\boldsymbol{y} - \boldsymbol{c}_l \Delta t, t) - \frac{\Delta t}{\tau} \left(f_l(\boldsymbol{y} - \boldsymbol{c}_l \Delta t, t) - f_l^{(eq)} \right) \tag{8}$$

one easily identifies the overall structure of the computation that evolves the system by one time step Δt; for each point $\boldsymbol{y}$ in the discrete grid one:

1. gathers from neighboring sites the values of the fields f_l corresponding to populations that drift towards y, and then
2. performs all mathematical processing needed to compute (*on a point-by-point* basis) the quantities appearing in the equation above.

The key remark is that both steps above are completely uncorrelated for different points of the grid, so they can be parallelized according to any convenient scheme, as long as step (1) is done before step (2).

These features are well suited for emerging HPC architectures, that foresee a large number of processing nodes, each made by many processing-cores, and the cores themselves able to perform SIMD operations. The challenge then rests in matching algorithm-level parallelism with all available computing resources.

Further help comes from the simple node-to-node communication pattern required by the LB method when implemented on a multi-node system. The easiest approach is that of tiling the physical grid on the set of available processors, so nearest-neighbor data moves translate to communications between nearest-neighbor processors assembled in a torus topology. Finally, note that *all* remote communications can be done at just one specific point in the main iteration loop, before population data is gathered for processing.

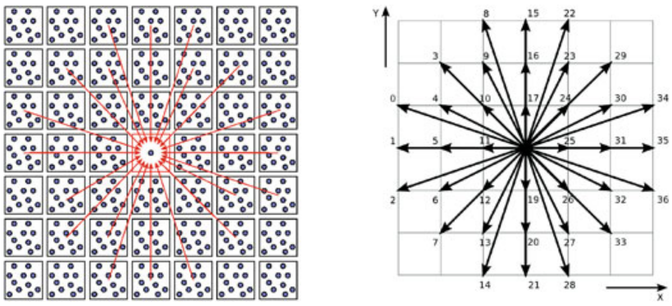


Fig. 1. Velocity vectors ($\boldsymbol{c}_l$) for the LB populations of the D2Q37 model. Population labelling is arbitrary; populations associated to each site are stored at consecutive memory address in the order defined by their labels.

At each time-step the update algorithm processes each lattice-point by applying in sequence two computationally relevant kernels, *propagate* and *collide*. Kernel *propagate* moves populations across the lattice points according to the pattern shown in Fig. 1. For each site, it updates the populations associated to it, gathering the new values from 36 neighbor sites at distance up to 3 in the physical lattice. The load of this kernel is dominated by memory copies at sparse memory addresses. The other key kernel, *collide*, computes the collisional operator on the population values gathered in the previous phase. This kernel performs all mathematical steps associated to Eq. 8, operating on the populations sitting at each site. This is the most floating-point intensive part of the code, performing around 7660 double-precision operations per site.

3 The Blue Gene/Q System

The *Blue Gene/Q* (BG/Q) machine is the latest generation massively parallel systems of the Blue Gene family developed by IBM.

BG/Q is a five-dimensional toroidal grid of nodes. Each node has one A2 processor, running at 1.6 GHz. It contains 18 cores; 16 cores are used by the application program, one by the operating system, and one is redundant. Each core is a 4-way multi-threaded CPU, executing up to 4 independent threads. Each core has 16(D) + 16(I) KB of L1-cache, and a SIMD floating point unit; the latter executes QPX vector instructions, that compute 4 double-precision operations in parallel. All in all, the node has a peak performance of 204.8 GFlops. All cores share a 32 MB L2-cache; external memory is 16 GB, handled by two memory controllers; the effective (read or write) bandwidth is close to 30 GB/s. Finally, the processor includes an interface to the 5D torus network, based on ten duplex links of 2 GB/s each. See [9] for more details.

Each node runs an instance of *Compute Node Kernel* (CNK), a lightweight version of the Linux kernel. The machine can be programmed using a standard MPI approach, handling parallelism both among and within the nodes. Alternatively, intra-node parallelism can be managed via *openMP* or *PThread* libraries.

Performance on BG/Q heavily relies on the ability of programmers, compilers and run-time support to carefully exploit parallelism at *all* levels. Mapping the code in a balanced way on a large number of processing elements is relatively straightforward in our case; however, more delicate issues relate to:

- core parallelism: the code should allow all cores to work in parallel exploiting either breaking down the application is in several sub-tasks, each run by a different core, or partitioning the data-set among the cores, each executing the same task on different data items.
- vector programming: each core processes the data-set of the application using QPX vector instructions and exploiting streaming-parallelism (SIMD);

SIMD optimizations can be applied either by the compiler, or by the programmer by coding with *intrinsic* functions. In the first case the program is a scalar code (all variables are scalar) and the compiler – if it detects specific conditions – automatically inserts SIMD streaming instructions. For example, if no data dependencies occur between iterations in a loop, the compiler can (partially) unroll it, so two or more iterations are processed in parallel by vector instructions. This easily applied approach is limited in performance by the ability of the compiler to identify parallelizable structures. In an often more efficient approach, the programmer codes the algorithm using vector variables, processed by so called *intrinsic* functions which directly map onto vector assembly instructions. For example, a double precision fused-multiply-add on vectors of 4 double-precision elements reads `d = vec_madd (a, b, c)`, where `a, b, c, d` are vector variables of type `vector4double`.

4 Implementation of the D2Q37 Model

In this section we describe the structure of our BG/Q optimized D2Q37 LB code, its data-structures, implementation and optimization details.

We consider a physical lattice in 2 dimensions of size $L_x \times L_y$. We split it on N nodes along the X dimension: each node hosts a sub-lattice of $L_x/N \times L_y$ sites. One could consider a different mapping, e.g. splitting along Y; however, since we plan to use our code for physics simulations with many aspect-ratios (both $L_x > L_y$ and $L_x < L_y$) and communication overheads are small (see later for details) we arbitrarily pick one of the two options. This mapping implies that nodes are virtually arranged at the edges of a ring, so each node is connected with a previous and a next one. This connectivity can be easily and efficiently mapped on the 5D-torus of BG/Q. Communications use the MPI library.

The physical variables associated to each lattice site are 37 double precision floating-point values, representing the populations of the model. On each node, populations are stored in column-major order, as this makes node-to-node communications more efficient (see later). We keep two copies of the lattice in memory. Each step of the algorithm reads one copy and updates the other, making the implementation of the code much simpler. For 2D systems memory is not a critical resource, and the memory footprint required to store our lattices is not too large. On each node we also allocate three *halo*-columns, at the right and left edges of the sub-lattice belonging to that node. Population data coming from the three adjoining physical columns of the neighbor nodes is copied to the *halo*s in one node-to-node communication phase, at the beginning of the *propagate* phase. Once this is done, all remaining steps are local to each node so they run independently and concurrently.

The lattice is arranged in memory according as an *Array of Structure* (AoS): population members of a site are stored one after the other. This improves the locality of populations associated to each site, and better suits the cache structure, see [3], improving the performance of the *collide* kernel.

Our code tries to exploit all parallelism opportunities: the simulation is organized as a multi-node program, on each node we use a multi-threaded program and, last but not least, each thread uses vector instructions.

To exploit-core parallelism within each node the sub-lattice is further sliced along the X dimension, and each slice is processed by a different thread. Parallelism within the core is managed using the openMP library: we configure the program to run 1 up-to 4 threads per core.

The assignment of slices to threads is different at different stages of the program: when the program executes the *collide* kernel, all slices have the same size (since the work load is the same at all lattice sites). Things are different for the *propagate* kernel: in this phase one must first perform a node-to-node communication to retrieve from neighboring nodes (and store into the halos) the data items needed to process the two sets of three columns laying close to the sub-lattice edges; remote data are however not needed to process all other lattice points. We therefore split the sub-lattice of each node among the N_t threads running on that node: the first and the last three columns are managed by threads 0 and $N_t - 1$, and all other columns are evenly split on the other threads. Threads 0 and $N_t - 1$ need to perform node-to-node communications, but they have a much smaller computational load: in this way we almost fully hide communication overheads.

We structure the code of each thread as follows (see also Fig. 2):

1. threads 0 and $N_t - 1$ perform MPI send-receives to update halos with neighbor nodes; they then execute *propagate* on three columns each;
2. concurrently with step 1, the remaining threads run *propagate* in parallel on their lattice slices;
3. threads synchronize at the end of steps 1 and 2. They then execute the *bc* kernel on the three rows at the top and bottom of the lattice, enforcing boundary conditions (e.g., a constant temperature and zero velocity), by adjusting the population variables close to the top and bottom of the lattice;
4. finally, all threads start computing the *collide* kernel, each on a different (and equally sized) slice of the lattice.

Within each thread, we introduce a further level of parallelism and process several lattice-sites in parallel: we use QPX streaming instructions acting on vectors gathering population variables from 4 sites. We have handcrafted our

```
for ( step = 0; step < MAXSTEP; step++ ) {
  #pragma omp barrier
  if ( tid == 0 || tid == N_t - 1 ) {
    MPI_Sendrecv ( ... ); // update halos
    propagate(); // apply propagate() to left- and right-halos
  } else {
    propagate(); // apply propagate() to inner part
  }
  #pragma omp barrier
  bc ( TOP );     // apply bc() to three upper row-sites
  bc ( LOW );     // apply bc() to three lower row-sites
  #pragma omp barrier
  collide( ... ); // apply collide()
}
```

Fig. 2. Code executed by each thread. The various phases are executed by a variable number of threads, and are synchronized through barriers.

vectorized routines, using intrinsic functions. We divide the lattice in 4 parts along the Y dimension, and we pack together populations of sites at distance $L_y/4$ on one 256-bit QPX vector. We then process together 4 lattice sites, using a mix of operations which is the same for all sites packed on the same QPX vector.

5 Performance Analysis

In this section we present and comment on performance results. We focus on the two main kernels – *propagate* and *collide* – since all other steps in the program have a truly negligible impact on running time.

We start by analyzing single-node performance; our results are shown in Fig. 3. We see that there is roughly a factor $2X$ in performance between the handcrafted vectorized version of the code and the one automatically vectorized by the compiler: the latter is probably not able to spot all opportunities for vectorization; so, while this result may be encouraging for an automatic tool, if performance is top priority, human intervention on the code is still necessary. Performance increases significantly as the number of threads grows for the *collide* kernel. This is expected, since increasing the number of threads run by one core helps hides latency between dependent operations in this strongly compute-bound kernel. This is less evident for *propagate*, as in this case the performance bottleneck is associated to complex address patterns and to short bursts of memory accesses. All in all the node delivers approximately 30 % of its peak performance, using some 75 % of its memory bandwidth.

The performance of the full production-grade code as a function of the number of cores in shown in Fig. 4 in a strong-scaling regime for two lattice sizes, namely 16384 × 8192 and 32768 × 1024. Our first choice is a typical size for

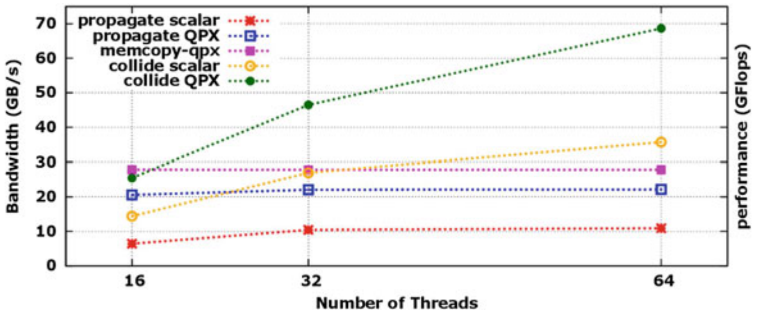


Fig. 3. Bandwidth of the *propagate* kernel (left vertical scale) and floating-point performance of the *collide* kernel (right vertical scale) as a function of the number of threads on a lattice of 1024 × 8192 sites running on one BG/Q node. We compare scalar and QPX-enabled codes. We also include for reference the bandwidth performance of a memory-copy benchmark.

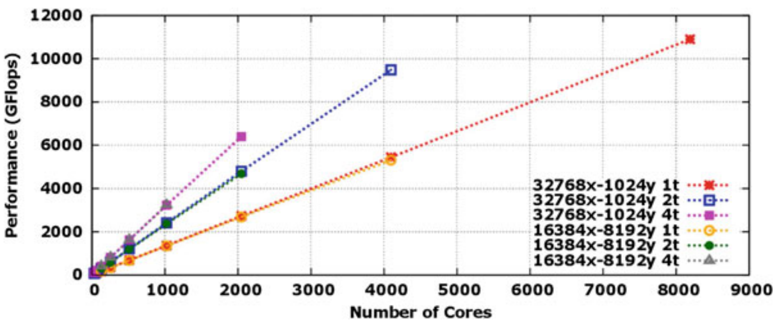


Fig. 4. Performance of the full (production-grade) code for two lattice sizes as a function of the number of processing cores. We plot performance for runs using 1, 2 and 4 threads per core.

a state-of-the-art simulation, while our second choice has a somewhat awkward form-factor that we have picked to try to expose latency effects in the node-to-node communication structure. The picture shows excellent scaling figures, clearly stretching to the largest number of cores (or threads) on which the physical system can be mapped.

Figure 5 analyzes the reasons behind the scaling figures seen above, by plotting – again as a function of the number of cores – the (normalized) ratio of performance over number of cores. This ratio should be constant for perfect strong scaling; we see that this is clearly so for the *collide* kernel, while the *propagate* kernel shows clear (albeit limited) evidence for super-scaling.

We find a clean explanation of this behavior in Fig. 6, where we plot histograms of the execution times of the *propagate* kernel; we distinguish between the threads handling lattice columns close to the halo (these threads perform a node-to-node communications) and the threads (handling the bulk of the lattice)

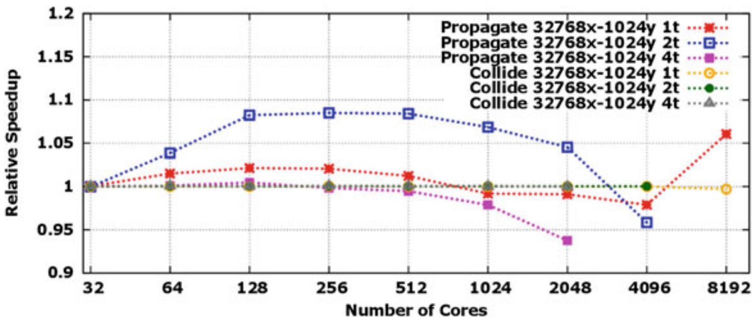


Fig. 5. Ratio of speed-up over number of cores for the *propagate* and *collide* kernels, as a function of the number of cores. We plot values for runs using 1, 2 and 4 threads per core. This ratio equals 1 for perfect scaling.

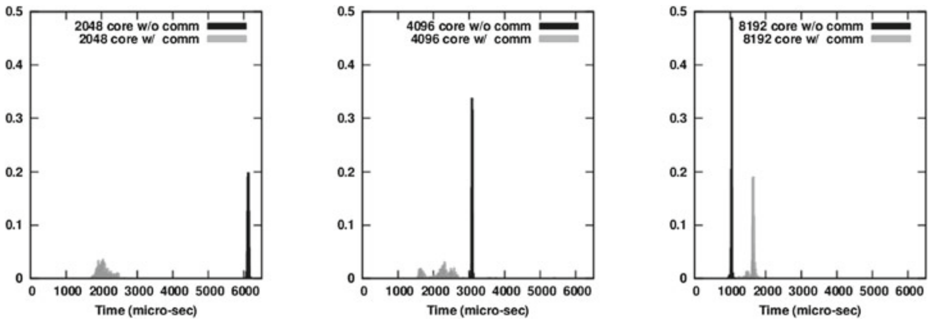


Fig. 6. Histograms of the *propagate* kernel execution-time for runs with three different numbers of cores (2048, 4096, 8192), using one thread per core. For each run, we build separate histograms for threads requiring node-to-node communications (w/ comm) and for threads without communications (w/o comm); for easy comparison we use the same scales for all the three panels.

that do not perform communications. In spite of a large variance of the measurements, one clearly sees that (i) the threads that contain a communication step are faster than the others (except when the largest number of cores is used), and (ii) the execution time of threads not performing communications scales much faster than perfect scaling would imply. Point (i) above shows that we have hidden communication latency almost perfectly in the application, while point (ii) is due to cache effects in this strongly memory-bound routine, that become more favorable as the lattice handled by each core gets smaller.

6 Conclusions

In this paper we have described the details of an optimized implementation of a state-of-the-art LB code. The program has been structured in such a way to match the parallel structure of the algorithm with the parallel features of the target computing system. Once an appropriate "coarse-grained" structure of the code and of the mapping of the variables on the node have been defined, the lowest level of parallelism (SIMD vectorization) can be automatically exploited by the compiler. However, if performance is top priority, also this final step must be handled by a programmer.

Performance results at the one node level are not as good as those reported for GPUs (Ref. [5] quotes $\simeq$130 Gflops per GPU), which results the best option as long as the lattice size fits the available amount of memory ($\leq$8 GB). Simulation of larger lattices must be split over a several GPUs, and in this situation, superior network integration makes the BG/Q implementation a better option in terms of strong-scaling.

Acknowledgements. We would like to thank CINECA (Bologna, Italy) and the Jülich Supercomputing Center (Jülich, Germany) for access to the *Fermi* and *Juqueen* BG/Q systems. This work has been done in the framework of the *COKA* and *SUMA* projects of INFN (Italy).

References

1. Succi, S.: The Lattice Boltzmann Equation for Fluid Dynamics and Beyond. Oxford University Press, Oxford (2001)
2. Bartoloni, A., et al.: LBE simulations of Rayleigh-Benard convection on the APE100 parallel processor. Int. J. Mod. Phys. **C4**, 993 (1993)
3. Pohl, T., et al.: Optimization and profiling of the cache performance of parallel lattice Boltzmann codes. Parallel Process. Lett. **13**(4), 549 (2003)
4. Wellein, G., Zeiser, T., Hager, G., Donath, S.: On the single processor performance of simple lattice Boltzmann kernels. Comput. Fluids **35**, 910 (2006)
5. Biferale, L., et al.: A multi-GPU implementation of a D2Q37 lattice Boltzmann code. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011, Part I. LNCS, vol. 7203, pp. 640–650. Springer, Heidelberg (2012)
6. Biferale, L., et al.: Optimization of multi-phase compressible lattice Boltzmann codes on massively parallel multi-core systems. Procedia Comput. Sci. **4**, 994–1003 (2011)
7. Sbragaglia, M., et al.: Lattice Boltzmann method with self-consistent thermo-hydrodynamic equilibria. J. Fluid Mech. **628**, 299 (2009)
8. Scagliarini, A., et al.: Lattice Boltzmann methods for thermal flows: continuum limit and applications to compressible Rayleigh-Taylor systems. Phys. Fluids **22**, 055101 (2010)
9. Chen, D., et al.: The IBM Blue Gene/Q interconnection network and message unit. In: Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, vol. 26 (2011)

A Parallel and Scalable Iterative Solver for Sequences of Dense Eigenproblems Arising in FLAPW

Mario Berljafa[1] and Edoardo Di Napoli[2,3(✉)]

[1] School of Mathematics, The University of Manchester, Alan Turing Building, Manchester M13 9PL, UK
m.berljafa@maths.man.ac.uk

[2] Jülich Supercomputing Centre, Forschungszentrum Jülich, Wilhelm-Johnen straße, 52425 Jülich, Germany
e.di.napoli@fz-juelich.de

[3] Aachen Institute for Advance Study in Computational Engineering Science, Schinkelstraße 2, 52062 Aachen, Germany
dinapoli@aices.rwth-aachen.de

Abstract. In one of the most important methods in Density Functional Theory – the Full-Potential Linearized Augmented Plane Wave (FLAPW) method – dense generalized eigenproblems are organized in long sequences. Moreover each eigenproblem is strongly correlated to the next one in the sequence. We propose a novel approach which exploits such correlation through the use of an eigensolver based on subspace iteration and accelerated with Chebyshev polynomials. The resulting solver, parallelized using the Elemental library framework, achieves excellent scalability and is competitive with current dense parallel eigensolvers.

Keywords: Chebyshev polynomials · Subspace iteration · Eigenproblem sequence · Density functional theory · Elemental

1 Introduction

We present a methodological approach to solve for eigenpairs of sequences of correlated dense eigenproblems arising in Density Functional Theory (DFT). The novelty of this approach resides in the use of approximate solutions in combination with a simple block eigensolver based on polynomially accelerated subspace iteration. When parallelized for distributed memory architectures this iterative method is a viable alternative to conventional dense eigensolvers both in terms of scalability and performance. Ultimately our approach will enable the DFT specialists to simulate larger and more complex physical systems.

Within the realm of condensed-matter physics, DFT is considered the standard model to run accurate simulations of materials. The importance of these simulations is two-fold: on the one hand they are used to verify the correctness of the quantum mechanical interpretation of existing materials. On the other hand, simulations constitute an extraordinary tool to verify the validity of new atomistic models which may ultimately lead to the invention of brand new materials.

Each simulation consists of a series of self-consistent field (SCF) cycles; within each cycle a fixed number $\mathcal{N}_{\mathbf{k}}$ of independent eigenvalue problems is solved. Since dozens of cycles are necessary to complete one simulation, one ends up with $\mathcal{N}_{\mathbf{k}}$ sequences made of dozens of eigenproblems. The properties of these eigenproblems depend on the discretization strategy of the specific DFT method of choice. In this paper we will exclusively consider the Full-Potential Linearized Augmented Plane Waves method (FLAPW). This DFT method gives rise to dense hermitian generalized eigenproblems (DGEVP) with matrix size typically ranging from 2,000 to 20,000.

In FLAPW only a fraction of the lowest part of the eigenspectrum is required. The eigenvalues inside this fraction correspond to the energy levels below Fermi energy and their number never falls below 3 % or exceeds 20 % of the eigenspectrum. The relatively high number of eigenpairs in combination with the dense nature and the size of the eigenproblems inevitably lead to the choice of direct eigensolvers. Direct eigensolvers follow a constrained path of linear transformations starting from the generalized eigenproblem and arriving to a tridiagonal one. In turn, the tridiagonal problem is solved iteratively using one of the two methods available for computing just a fraction of the spectrum, namely bisection inverse iteration (BXINV) [1] and multiple relatively robust representations (MRRR) [2,3].

Until very recently, the computational strategy on parallel distributed memory architecture favored the use of ScaLAPACK [4] implementation of BXINV. Modern and efficient dense libraries, like ELPA [5] and EleMRRR [6], improve the performance but do not change the overall computational strategy: each problem in the sequence is solved in complete independence from the previous one. The latter choice is based on the view that problems in the sequence are considered only loosely connected. In fact despite the solution of one problem ultimately determine the initialization of the next, it does so in such a mathematically indirect manner that two successive problems seem to be quite independent. In this paper we propose a completely different strategy which tries to maximally exploit the sequence of eigenproblems using an iterative eigensolver as opposed to a direct one.

The originality of our approach, in spite of the assumed loose connection between eigenproblems, is in the use of the solutions of one problem in the sequence as input when solving the next one. By its inherent nature only an iterative method would be able to accept eigenvectors as input. On the other hand not all such methods are capable of maximally exploiting the information inputed. In this regards one of the most effective methods is Subspace Iteration (SI). We have implemented a version of this method accelerated with Chebyshev polynomials. The end result is an algorithm (ChFSI) whose bulk of computations is performed making use of the highly optimized Basic Linear Algebra Subroutines (BLAS) library and can be easily parallelized on shared and distributed memory architectures. In this paper we present preliminary results for a distributed memory version of ChFSI implemented using the Elemental library framework [7].

2 FLAPW Simulations on Large Parallel Architectures

Every DFT method is based on a variational principle stemming from the fundamental work of Kohn and Hohenberg [8], and its practical realization [9]. Central to DFT is the solution of a large number of coupled one-particle Schrödinger-like equations known as Kohn-Sham (KS).

$$\left(\frac{\hbar^2}{2m}\nabla^2 + \mathcal{V}_{\text{eff}}[n(\mathbf{r})]\right)\phi_i(\mathbf{r}) = E_i\phi_i(\mathbf{r}) \quad ; \quad n(\mathbf{r}) = \sum_i f_i \phi_i(\mathbf{r})$$

Due to the dependence of the effective potential $\mathcal{V}_{\text{eff}}$ on the charge density $n(\mathbf{r})$, in itself a function of the orbital wave functions $\phi_i(\mathbf{r})$, the KS equations are non-linear and are generally solved self-consistently.

The KS equations need to be "discretized" in order to be solved numerically. Intended in its broadest numerical sense, the discretization translates the KS equations in a non-linear eigenvalue problem. Eigenproblems generated by distinct discretization schemes have numerical properties that are often significantly different; for sake of simplicity we can group most of the schemes in three classes. The first and the second classes make respectively use of plane waves and localized functions to expand the one-particle orbital wave functions $\phi_i(\mathbf{r})$ appearing in the KS equations

$$\phi_i(\mathbf{r}) \longrightarrow \phi_{\mathbf{k},i}(\mathbf{r}) = \sum_{\mathbf{G}} c^{\mathbf{G}}_{\mathbf{k},i} \psi_{\mathbf{G}}(\mathbf{k},\mathbf{r}). \tag{1}$$

Methods in the third class do not use an explicit basis for the $\phi_i(\mathbf{r})$'s but discretize the KS equations on a grid in real space using finite differences.

The eigenvalue problems emerging from the first two discretization classes consist of dense matrices of small-to-moderate size while, within real space methods, one ends up with very large sparse matrices. Due to the dramatically different set of properties of the eigenproblems, each DFT method uses a distinct strategy in solving for the required eigenpairs. For instance it is quite common that methods based on plane waves (ABINIT, VASP, PARATEC, Castep, ...) use direct eigensolvers while real space methods (PARSEC, GPAW, Octopus, ...) make use of iterative eigensolver based on Krylov- or Davidson-like subspace construction. From the point of view of software packages for distributed memory architectures, the choice between direct or iterative eigensolvers leads respectively to the use of traditional parallel libraries like ScaLAPACK or PARPACK [10].

In this paper we deal with a specific instance of a plane wave method which splits the basis functions support domain: in a spherical symmetric area around each atom, $\psi_{\mathbf{G}}$ receive contributions by augmented radial functions, while plane waves are supported in the interstitial space between atoms. This discretization of the KS equations – known as FLAPW – translates in a set of $\mathcal{N}_{\mathbf{k}}$ quite dense DGEVPs

$$\sum_{\mathbf{G}'}(A_{\mathbf{k}})_{\mathbf{G}\mathbf{G}'}\ c_{\mathbf{k},i}^{\mathbf{G}'} = \lambda_{\mathbf{k},i}\sum_{\mathbf{G}'}(B_{\mathbf{k}})_{\mathbf{G}\mathbf{G}'}\ c_{\mathbf{k},i}^{\mathbf{G}'},$$

each one labeled by a value of the plane wave vector $\mathbf{k}$. The role of eigenvectors is played by the n-tuple of coefficients $c_{\mathbf{k},i}$ expressing the orbital wave functions ϕ_i in terms of the basis wave functions $\psi_{\mathbf{G}}$.

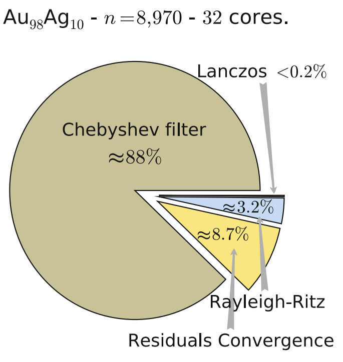
The entries of each DGEVP matrix are initialized by evaluating numerically a series of expensive multiple integrals involving the $\psi_{\mathbf{G}}$s. Since we are dealing with non-linear eigenvalue problems, each DGEVP has to be solved in a chain of SCF-cycles labeled by ℓ

$$A_{\mathbf{k}}\ c_{\mathbf{k},i} = \lambda_{\mathbf{k},i} B_{\mathbf{k}}\ c_{\mathbf{k},i} \ \longrightarrow\ P_{\mathbf{k}}^{(\ell)}:\ A_{\mathbf{k}}^{(\ell)}\ c_{\mathbf{k},i}^{(\ell)} = \lambda_{\mathbf{k},i}^{(\ell)} B_{\mathbf{k}}^{(\ell)}\ c_{\mathbf{k},i}^{(\ell)} \qquad (\ell = 1,\dots,N).$$

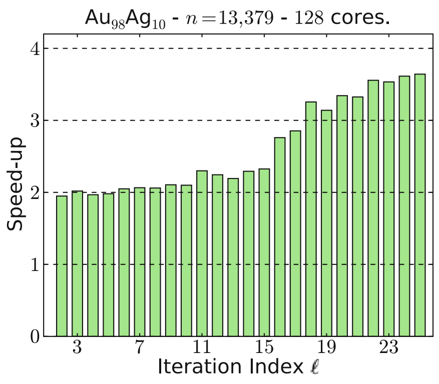
All along the sequence the solutions of all $P_{\mathbf{k}}^{(\ell-1)}$ are used to initialize the new eigenproblems $P_{\mathbf{k}}^{(\ell)}$. In particular the eigenvectors $c_{\mathbf{k},i}^{(\ell-1)}$ are used to derive the orbital functions $\phi_{\mathbf{k},i}^{(\ell-1)}$ which in turn contribute to the charge density $n^{(\ell-1)}(\mathbf{r})$. At the next cycle $n^{(\ell-1)}(\mathbf{r})$ contributes to modify the potential $\mathcal{V}_{\text{eff}}$ which causes the functional form of the $\psi_{\mathbf{G}}^{(\ell)}$s to change. These new basis function set directly determines the initialization of the entries of $A_{\mathbf{k}}^{(\ell)}$ and $B_{\mathbf{k}}^{(\ell)}$ and indirectly the new eigenvectors $c_{\mathbf{k},i}^{(\ell)}$. The result are a number $\mathcal{N}_{\mathbf{k}}$ of sequences of eigenproblems $\left\{P_{\mathbf{k}}^{(1)} \dots P_{\mathbf{k}}^{(N)}\right\}$, one sequence for each fixed $\mathbf{k}$, where the eigenpairs $(\lambda_{\mathbf{k},i}^{(N)}, c_{\mathbf{k},i}^{(N)})$ converged within tolerance to the solution of the original non-linear problem.

In theory the chain of computations that goes from $P_{\mathbf{k}}^{(\ell-1)}$ to $P_{\mathbf{k}}^{(\ell)}$ implies a connection between eigenvectors of successive eigenproblems. The entries of $P_{\mathbf{k}}^{(\ell)}$ are in fact the result of multiple integrals between $\psi_{\mathbf{G}}^{(\ell)}$ and operators depending on the new charge density $n^{(\ell-1)}(\mathbf{r})$. All these quantities are modified by $c_{\mathbf{k},i}^{(\ell-1)}$ in a distinct non-linear fashion. Consequently there is no known mathematical formulation which makes this connection explicit. Correlation between the eigenvectors becomes evident only numerically [11].

When solving for an eigenvalue problem the first high level choice is between direct and iterative eigensolvers. The first are in general used to solve for a large portion of the eigenspectrum of dense problems. The latter are instead the typical choice for sparse eigenproblems or used to solve for just few eigenpairs of dense ones. In FLAPW the hermitian matrices $A_{\mathbf{k}}$ and $B_{\mathbf{k}}$ are quite dense, have size not exceeding 20,000, and each $P_{\mathbf{k}}^{(\ell)}$ is solved for a portion of the lower spectrum not bigger than 20 %. Consequently, when each DGEVP is singled out from the rest of the sequence, direct solvers are unquestionably the method of



(a) Distribution of computing time.



(b) Random vs Approximate vectors.

Fig. 1. The data in this figure refers to eigenproblems of distinct sizes n relative to the same physical system $Au_{98}Ag_{10}$. Plot (a) represents the computing fractions of EleChFSI's main algorithmic steps w.r.t. the total computing time. Plot (b) shows the speed-up of EleChFSI when inputed approximate solutions as opposed to random vectors.

choice. Currently, most of the codes based on FLAPW methods [12–14] use the algorithms BXINV or MRRR directly out of the ScaLAPACK or ELPA library.

If the use of direct solvers is the obvious choice when each $P_{\mathbf{k}}^{(\ell)}$ is solved in isolation, the same conclusion may not be drawn when we look at the entire sequence of $\left\{P_{\mathbf{k}}^{(\ell)}\right\}$. In [11] it is shown how the correlation between eigenvectors of successive DGEVPs becomes manifest in the evolution of the angles $\theta_{\mathbf{k},i}^{(\ell)} = \langle c_{\mathbf{k},i}^{(\ell-1)}, c_{\mathbf{k},i}^{(\ell)}\rangle$. In particular the $\theta_{\mathbf{k},i}^{(\ell)}$ decrease almost monotonically as a function of cycle index ℓ, going from $\sim 10^{-1}$ down to $\sim 10^{-8}$ towards the end of the sequence.

The empirical evolution of the eigenvectors suggests that they can be "reused" as approximate solutions, and inputed to the eigensolver at the successive cycle. Unfortunately no direct eigensolver is capable of accepting vectors as approximate solutions. Therefore if we want to exploit the progressive collinearity of vectors as the sequence progresses, we are lead to consider iterative solvers; these solvers by their own nature build approximate eigenspaces by manipulating approximate eigenvectors. In particular we need a block iterative eigensolver that accepts at the same time many vectors as input. Among the many choices of block solvers, the Chebyshev Filtered Subspace Iteration method (ChFSI) showed the highest potential to take advantage of approximate eigenvectors [15] (see also Fig. 1(b)). Since the core of the algorithm is based on the repetitive use of matrix-matrix multiplications, the use of the BLAS 3 library makes it very efficient and easy to scale.

3 The Parallel Chebyshev Subspace Iteration

Subspace Iteration complemented with a Chebyshev polynomial filter is a well known algorithm in the literature [16]. A version of it was recently developed for a real space discretization of DFT by Chelikowsky *et al.* [17,18] and included in the PARSEC code [19].

SI is probably one of the earliest iterative algorithms to be used as numerical eigensolver. It is by definition a block solver since it simply attempts to build an invariant eigenspace by multiplying a block of vectors with the operator to be diagonalized. It is a known fact that any implementation based on subspace iteration converges very slowly. By using a polynomial filter on the initial block of inputed vectors the method experiences a high rate of acceleration. Unfortunately the block of vectors spanning the invariant subspace could easily become linearly dependent. In order to avoid such an occurrence SI is usually complemented with some re-orthogonalization procedure.

Algorithm 1. Chebyshev Filtered Subspace Iteration with locking

Input: Matrix $H^{(\ell)}$ of the DGEVP reduced to standard form, approximate eigenvectors $\hat{Y}^{(\ell-1)} \overset{\text{def}}{=} \left[\hat{y}_1^{(\ell-1)}, \ldots, \hat{y}_{\text{NEV}}^{(\ell-1)}\right]$ and eigenvalues $\lambda_1^{(\ell-1)}$ and $\lambda_{\text{NEV}+1}^{(\ell-1)}$.
Output: Wanted eigenpairs (Λ, Y).

```
 1: Estimate the largest eigenvalue.                            ▷ Lanczos
 2: repeat
 3:     Filter the vectors, $\hat{Y} = C_m(\hat{Y})$.           ▷ Chebyshev filter
 4:     Re-orthonormalize $\hat{Y}$.                            ▷ QR algorithm
 5:     Compute Rayleigh quotient $G = \hat{Y}^\dagger H^{(\ell)} \hat{Y}$.  ▷ Rayleigh-Ritz (Start)
 6:     Solve the reduced problem $G\hat{W} = \hat{W}\hat{\Lambda}$.
 7:     Compute $\hat{Y} = \hat{Y}\hat{W}$.                     ▷ Rayleigh-Ritz (End)
 8:     for $i =$ converged $\rightarrow$ NEV do                ▷ Deflation & Locking (Start)
 9:         if Res$(\hat{Y}_{:,i}, \hat{\Lambda}_i) <$ TOL then
10:             $\Lambda = [\Lambda \ \hat{\Lambda}_i]$
11:             $Y = [Y \ \hat{Y}_{:,i}]$
12:         end if
13:     end for                                                 ▷ Deflation & Locking (End)
14: until converged $\geq$ NEV
```

Our ChFSI algorithm is a slightly more sophisticated version of the basic SI and is specifically tailored for DFT-like eigenproblems. The whole algorithm is illustrated in the **Algorithm 1** scheme. Notice that the initial input is not the initial $P^{(\ell)}$ but its reduction to standard form $H^{(\ell)} = L^{-1}A^{(\ell)}L^{-\mathrm{T}}$ where $B^{(\ell)} = LL^{\mathrm{T}}$, and $\hat{Y}^{(\ell-1)}$ are the eigenvectors of $H^{(\ell-1)}$. ChFSI uses few Lanczos iterations (`line 1`) so as to estimate the upper limit of the eigenproblem spectrum [20]. This estimate is necessary for the correct usage of the filter based on Chebyshev polynomials [16]. After the Chebyshev filter step (`line 3`) the resulting block of vectors is re-orthonormalized using a simple QR algorithm (`line 4`) followed by a Rayleigh-Ritz procedure (`line 5`). At the end of the Rayleigh-Ritz step eigenvector residuals are computed, converged eigenpairs are deflated and locked (`line 13`) while the non-converged vectors are sent again to the filter to repeat the whole procedure.

The Chebyshev polynomial filter is at the core of the algorithm. The vectors $\hat{Y}$ are filtered exploiting the 3-terms recurrence relation which defines Chebyshev polynomials of the first kind

$$C_{m+1}(\hat{Y}) = 2\ H\ C_m(\hat{Y}) - C_{m-1}(\hat{Y}) \quad ; \quad C_m(\hat{Y}) \overset{\text{def}}{=} C_m(H) \cdot \hat{Y}. \tag{2}$$

This construction implies all operations internal to the filter are executed through the use of ZGEMM, the most performant among BLAS 3 routines. Since roughly 90 % of the total CPU time is spent in the filter (see pie chart in Fig. 1), the massive use of ZGEMM makes ChFSI quite an efficient algorithm and potentially a very scalable one.

The parallel MPI version of ChFSI (EleChFSI) is implemented within the Elemental library, a framework for distributed memory dense linear algebra. The core of the library is the two-dimensional cyclic element-wise ("elemental" or "torus-wrap") matrix distribution (default distribution hereafter). The p MPI processes involved in the computation are logically viewed as a two-dimensional $r \times c$ process grid with $p = r \times c$. The matrix $A = [a_{ij}] \in \mathbb{F}^{n \times m}$ is distributed over the grid in such a way that the process (s, t) owns the matrix

$$A_{s,t} = \begin{pmatrix} a_{\gamma,\delta} & a_{\gamma,\delta+c} & \cdots \\ a_{\gamma+r,\delta} & a_{\gamma+r,\delta+c} & \cdots \\ \vdots & \vdots & \end{pmatrix},$$

where $\gamma \equiv (s + \sigma_r) \mod r$ and $\delta \equiv (t + \sigma_c) \mod c$, and σ_r and σ_c are arbitrarily chosen alignment parameters.

For a given number $p > 1$ of processors there are several possible choices for r and c forming different grid shapes $(r, c) \overset{\text{def}}{=} r \times c$. Since the grid shape can have a significant impact on the overall performance, careful experiments should be undertaken in order to determine the best choice of (r, c). Another parameter which affects performance is the algorithmic block size. This term refers to the size of blocks of input data and is correlated to the square root of the L2 cache [21]. In practice, the effective size of the algorithmic block not only depends on the algorithm itself, but it is also affected by the architecture. Figure 2 shows that for EleChFSI a block size of 256 is always recommended independently of the number of cores or grid shape. This effect is imputable to the large number of matrix multiplications carried on by the filter.

In the EleChFSI algorithm the Hamiltonian and the approximate eigenvectors are distributed using the default distribution over the $r \times c$ grid employing the Elemental library `DistMatrix` class[1] which internally "hides" the details about the matrix data-type, size, leading dimension, and alignments. The net

[1] The library provides several other matrix distributions [7].

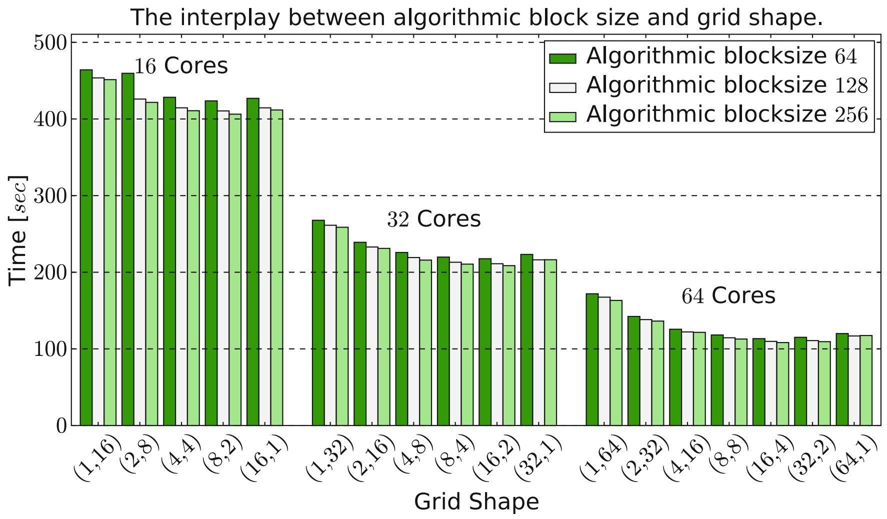


Fig. 2. The data in this plot refer to a DGEVP of $\ell = 20$, size $n = 13,379$, and number of sought after eigenpairs NEV = 972, corresponding to the physical system $\mathrm{Au_{98}Ag_{10}}$. The eigenproblem was repeatedly solved with EleChFSI using 16, 32, and 64 cores, all possible grid shapes (r, c) and three distinct algorithmic block sizes.

effect is to lift the user from the burden of passing all those attributes to internal routines as it is customary in (P)BLAS and (Sca/P)LAPACK libraries. The resulting separation of concerns allows for the parallelization of the Chebyshev filter in a straightforward fashion by calling the distributed memory implementation of ZGEMM. However, due to the generalization of the 3-term recursive relation, care must be taken with the distribution update of diagonal entries of the Hamiltonian.

The reduced eigenproblem in the Rayleigh-Ritz step is solved using a parallel implementation of the MRRR eigensolver – EleMRRR [6] – which is an integral part of Elemental. The deflation and locking mechanism deserves particular attention. When only a portion of the vectors are locked, the algorithm has to re-filter a number of vectors that may, in general, no longer have the same alignment σ_c. To overcome this problem the Elemental interface provides (among others) the routine `View`, which takes as arguments two distributed matrices A and B and four integers $i, j, height$ and $width$ and makes A a view of the $height \times width$ sub-matrix of B starting at coordinate (i, j).[2] The `View` routine works purely on pointers and fully handles the distribution details eliminating the need of allocating additional memory where to copy the data.

The communication for the computations is performed almost entirely in terms of collective communication within rows and columns of the process grid. Such strategy in general implies that a square grid shape is usually the best

[2] The function is overloaded, and there are thus other different definitions.

Table 1. Simulation data

Material	NEV	$\ell_{\max}$	n	Material	NEV	$\ell_{\max}$	n
		25	5,638			13	3,893
$Au_{98}Ag_{10}$	972	25	8,970	$Na_{15}Cl_{14}Li$	256	13	6,217
		25	13,379			13	9,273

option [22]. However, since in our case we are solving for a small fraction of the eigenspectrum, the matrix of vectors $\hat{Y}^{(\ell)}$ is tall and skinny. Consequently we expect that a narrow rectangular grid shape will do a better job than a square and wider one. This deduction is confirmed by Fig. 2; independently of the number of cores the optimal grid shape is either $(2^m, 4)$ or $(2^{m+1}, 2)$, where $m > 2$.

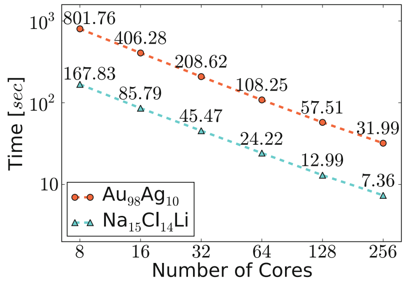
4 Numerical Results and Conclusions

The set of numerical tests presented here were performed on two distinct physical systems using three different sizes for the volume of the reciprocal space defining the range of the vector **G** appearing in (1). Consequently we obtained three sequences of eigenproblems for each physical systems[3]. The data of the sequences of eigenproblems are summarized in Table 1. All our numerical tests were performed on JUROPA, a large general purpose cluster where each node is equipped with 2 Intel Xeon X5570 (Nehalem-EP) quad-core processors at 2.9 GHz and 24 GB memory (DDR3, 1066 MHz). The nodes are connected by an Infiniband QDR network with a Fat-tree topology. The tested routines were compiled using the Intel compilers (ver. 12.0.3) with the flag -O3 and linked to the ParTec's ParaStation MPI library (ver. 5.0.26). The Elemental library (release 0.79) was used in conjunction with Intel's MKL BLAS (ver 11.0). All CPU times were measured by running each test multiple times and taking the average of the results. Eigenproblems were solved by EleChFSI by requiring the eigenpairs absolute residuals to be lower than 10^{-10}.

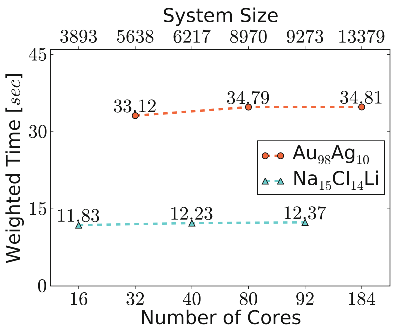
As already mentioned in the previous sections, Fig. 1 shows unequivocally the great advantage EleChFSI obtains from the use of the eigenvectors $\hat{Y}^{(\ell-1)}$ as input in solving the next eigenproblem $H^{(\ell)}$ in the sequence. This behavior is independent of the physical system or spectral properties of the eigenproblems: EleChFSI experiences speed-ups higher that 2X and often well above 3X towards the end of the sequence. Figure 2 also illustrate which is the optimal choice of grid shape and algorithmic block size. The remaining numerical tests were performed using exclusively the strategies outlined above.

Figure 3 illustrate the scalability, both strong and weak, of EleChFSI. Plot (a) shows a steady decrease of CPU time as the number of cores increases. The rate of reduction is practically the same for both systems despite their size differ

[3] We selected only sequences corresponding to the first **k** vector.



(a) EleChFSI's Strong scalability.



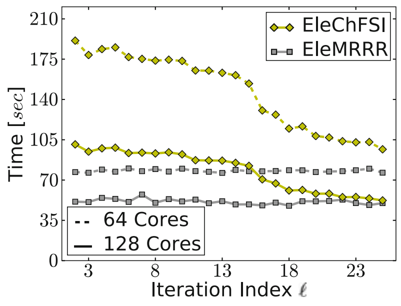
(b) EleChFSI's Weak scalability.

Fig. 3. EleChFSI's scalability for an increasing number of cores. In plot (a) the size of the eigenproblems are kept fixed while the number of cores is progressively increased. Eigenproblems of the two bigger system in Table 1 are tested, namely $n = 13,379$ and $n = 9,273$. In plot (b) all the systems are tested keeping the ratio of data per processor fixed. Times are weighted a posteriori by a factor keeping into account the ratio of operations per data varies in a non-predictable fashion with the size of the system.

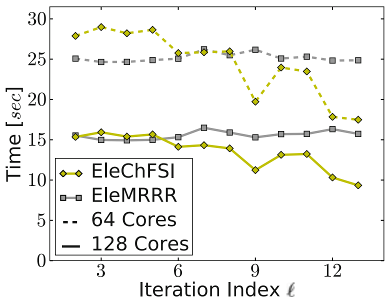
by more than 30 %. This plot shows that EleChFSI is extremely efficient even when the ratio of data per processor is not optimal. The weak scalability plot makes manifest the great potential of this algorithm for eigenproblems originating from FLAPW methods. The almost flatness of the two lines implies that large size eigenproblems can greatly exploit large supercomputing architectures. In other words EleChFSI has the potential of allowing the users of FLAPW-based codes to generate more complex physical systems made of thousands of atoms as opposed to just few hundreds.

Compared to direct solvers, EleChFSI promises to be quite competitive. Depending on the number of eigenpairs computed, our algorithm is on par or even faster than EleMRRR. In plot (a) of Fig. 4 EleChFSI appears to fall behind the direct solver when using just 64 cores. The situation improves substantially with 128 cores and at the end of the sequence both algorithms are on par. The situation is even more favorable in plot (b) where EleChFSI is already faster than EleMRRR for half of the eigenproblems in the sequence (64 cores). When the tests are repeated with 128 cores EleChFSI is distincly the faster of the two algorithms. Since the fraction of the spectrum computed in plot (a) and (b) is respectively ~7 % and ~3 %, Fig. 4 shows that EleChFSI scales better than EleMRRR and is more performant when the number of eigenpairs is not too high.

In conclusion, not only EleChFSI showed to take the greatest advantage from the progressive collinearity of eigenvectors along the sequence, but it proved to easily adapt to parallel architectures. We showed how such an algorithm, parallelized for distributed memory architectures, scales extremely well over a range of cores commensurate to the size of the eigenproblems. Compared to direct eigensolvers, EleChFSI is competitive with routines out of ScaLAPACK



(a) $Au_{98}Ag_{10}$ - $n = 13,379$



(b) $Na_{15}Cl_{14}Li$ - $n = 9,273$

Fig. 4. Comparing EleChFSI with EleMRRR on eigenproblems of increasing self-consistent cycle index ℓ. For the size of eigenproblems here tested the ScaLAPACK implementation of BXINV is comparable with EleMRRR [6]. For this reason a direct comparison with the BXINV solver is not included here.

and Elemental. Eventually the use of EleChFSI in FLAPW-based codes will enable the final user to access larger physical systems which are currently out of reach.

Acknowledgements. This research was in part supported by the VolkswagenStiftung through the fellowship "Computational Sciences". We are thankful to the Jülich Supercomputing Center for the computing time made available to perform the numerical tests. Special thanks to Daniel Wortmann and the FLEUR team for providing the input files that generated the eigenproblems used in the numerical tests.

References

1. Peters, G., Wilkinson, J.H.: The calculation of specified eigenvectors by inverse iteration. In: Bauer, F. (ed.) Linear Algebra, vol. 2 of Handbook for Automatic Computation, pp. 418–439. Springer, Berlin, Heidelberg (1971)
2. Dhillon, I.S.: A New O(n2) algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem, Ph.D. thesis, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA (1997)
3. Dhillon, I.S., Parlett, B.N.: Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices. Linear Algebra Appl. **387**, 1–28 (2004)
4. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK Users' Guide. SIAM, Philadelphia (1987)
5. Auckenthaler, T., Blum, V., Bungartz, H.J., Huckle, T., Johanni, R., Krämer, L., Lang, B., Lederer, H., Willems, P.R.: Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations. Parallel Comput. **37**(12), 783–794 (2011)
6. Petschow, M., Peise, E., Bientinesi, P.: High-performance solvers for dense Hermitian eigenproblems. SIAM J. Sci. Comput. **35**(1), c1–c22 (2013)

7. Poulson, J., Marker, B., van de Geijn, R.A., Hammond, J.R., Romero, N.A.: Elemental: a new framework for distributed memory dense matrix computations. ACM Trans. Math. Softw. **39**(2), 13:1–13:24 (2013)
8. Hohenberg, P.: Inhomogeneous electron gas. Phys. Rev. **136**(3B), B864–B871 (1964)
9. Kohn, W., Sham, L.J.: Self-consistent equations including exchange and correlation effects. Phys. Rev. **140**, A1133–A1138 (1965)
10. Lehoucq, R.B., Sorensen, D.C., Yang, C.: ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods. SIAM, Philadelphia (1998)
11. Di Napoli, E., Blügel, S., Bientinesi, P.: Correlations in sequences of generalized eigenproblems arising in density functional theory. Comput. Phys. Commun. **183**(8), 1674–1682 (2012)
12. Blügel, S., Bihlmayer, G., Wortmann, D.: FLEUR. http://www.flapw.de
13. Blaha, P., Schwarz, K., Madsen, G., Kvasnicka, D., Luitz, J.: Wien2k. http://www.wien2k.at/
14. Dewhurst, K., Sharma, S., Ambrosch-Draxl, C.: The exciting code. http://exciting-code.org/
15. Di Napoli, E., Berljafa, M.: Block iterative eigensolvers for sequences of correlated eigenvalue problems. Comput. Phys. Commun. **184**(11), 2478–2488 (2013)
16. Saad, Y.: Numerical methods for large eigenvalue problems. SIAM, Philadelphia (2011)
17. Zhou, Y., Saad, Y., Tiago, M.L., Chelikowsky, J.R.: Parallel self-consistent-field calculations via chebyshev-filtered subspace acceleration. Phys. Rev. E **74**(6), 066704 (2006)
18. Zhou, Y., Saad, Y., Tiago, M.L., Chelikowsky, J.R.: Self-consistent-field calculations using chebyshev-filtered subspace iteration. J. Comput. Phys. **219**(1), 172–184 (2006)
19. Kronik, L., Makmal, A., Tiago, M.L., Alemany, M.M.G., Jain, M., Huang, X., Saad, Y., Chelikowsky, J.R.: Parsec - the pseudopotential algorithm for real-space electronic structure calculations: recent advances and novel applications to nanostructures. Phys. Status Solidi B **243**(5), 1063–1079 (2006)
20. Zhou, Y., Li, R.C.: Bounding the spectrum of large hermitian matrices. Linear Algebra Appl. **435**(3), 480–493 (2011)
21. Goto, K., van de Geijn, R.A.: Anatomy of high-performance matrix multiplication. ACM Trans. Math. Softw. **34**(3), 1–25 (2008)
22. Chan, E., Heimlich, M., Purkayastha, A., van de Geijn, R.A.: Collective communication: theory, practice, and experience. Concurrency Comput. Pract. Exp. **19**(13), 1749–1783 (2007)

Sequential Monte Carlo in Bayesian Assessment of Contaminant Source Localization Based on the Sensors Concentration Measurements

Anna Wawrzynczak[1,2(✉)], Piotr Kopka[1,3], and Mieczyslaw Borysiewicz[1]

[1] National Centre for Nuclear Research, Świerk-Otwock, Poland
a.wawrzynczak@ncbj.gov.pl
[2] Institute of Computer Science, Siedlce University, Siedlce, Poland
[3] Poland Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Abstract. Accidental atmospheric releases of hazardous material pose great risks to human health and the environment. In this context it is valuable to develop the emergency action support system, which can quickly identify probable location and characteristics of the release source based on the measurement of dangerous substance concentration by the sensors network. In this context Bayesian approach occurs as a powerful tool being able to combine observed data along with prior knowledge to gain a current understanding of unknown model parameters.

We have applied the methodology combining Bayesian inference with Sequential Monte Carlo (SMC) to the problem of the atmospheric contaminant source localization. The algorithm input data are the on-line arriving concentrations of given substance registered by the distributed sensor's network.

We have proposed the different version of the Hybrid SMC along with Markov Chain Monte Carlo (MCMC) algorithms and examined its effectiveness to estimate the probabilistic distributions of atmospheric release parameters. The proposed algorithms scan 5-dimensional parameters' space searching for the contaminant source coordinates, release strength and atmospheric transport dispersion coefficients.

Keywords: Bayesian inference · Stochastic reconstruction · MCMC methods · SMC methods

1 Introduction

Environmental sensors have been deployed in various cities for early detection of contaminant releases into the atmosphere. Accidental atmospheric releases of hazardous material pose great risk to human health and the environment. During the event of an atmospheric release of chemical, radioactive or biological materials, emergency responders need to, as soon as possible, determine the location of source of dispersed substance. Such information help responders to make time-critical decisions regarding precautions for peoples safety, plans for evacuation and management of emergency services. In this context it is valuable to develop the emergency system which based on measurements of the concentration of dangerous substance by the network of sensors can inform about probable location of the release source. Moreover, the contamination source's location should be found as soon as possible.

It is clear that knowing gas source and wind field we can calculate the expected gas concentration for any downwind location. On the other hand, given concentration measurements and knowledge of the wind field and other atmospheric air parameters, finding the location of the source and its parameters is ambiguous. The problem has no unique solution and can be considered only in the probabilistic frameworks. To create the model realistically reflecting the real situation based only on a sparse point-concentration data is not trivial. This task requires specification of set of model's parameters. In the framework of Bayesian statistics all quantities included in the mathematical model are modeled as random variables with joint probability distributions. This randomness can be interpreted as parameter variability, and is reflected in the uncertainty of the true values expressed in terms of probability distributions. Bayesian methods reformulate the problem into searching for a solution based on efficient sampling of an ensemble of simulations, guided by comparisons with data. This methodology provides probabilistic estimates of parameters, which in turn are used to produce a forward dispersion model, which is necessary to obtain data to compare with real observations e.g.[1,2].

Previously [3,4] we have applied the methodology combining Bayesian inference with Marcov Chain Monte Carlo (MCMC) methods to the problem of the contaminant source localization. In this paper we propose the application of the Sequential Monte Carlo (SMC) methods combined with the Bayesian inference to the problem of the localization of the atmospheric contamination source. We present the possibility to connect MCMC and SMC to provide additional benefit in the process of event reconstruction. Proposed algorithms were tested on the synthetic release experiment.

2 Problem Setup

2.1 Synthetic Data

Our main goal is to conduct dynamic inference on an unknown atmospheric release and its impact as more information (data) accumulates. To test the proposed methods we require some concentration data. To satisfy this requirement we have performed the simulation with use of the atmospheric dispersion second-order Closure Integrated PUFF Model (SCIPUFF). SCIPUFF is an ensemble mean dispersion model designed to compute the time-dependent field of expected concentrations resulting from one or more sources. The model solves the transport equations using a second-order closure scheme and treats releases as a collection of Gaussian puffs [5]. In simulation we assumed that we have 10 sensors distributed over 15 km × 15 km area, the location of sensors was chosen randomly within the domain (Fig. 1). The contamination source was located at x = 2 km, y = 8 km, z = 50 m within the domain, the simulated release was continuous with rate q = 500 g/s and started one hour before first sensors measurements. The wind was directed along x axis with speed 5 m/s. Further in this paper we assume that the only algorithm input information we have, are reported every 15 min (in subsequent time steps) during 1.5 h concentrations of dispersed substance registered by 10 sensors (Fig. 1). We run algorithms searching for the source of contamination just after first information from sensors (time step = 1) and update the obtained probabilities with use of the developed algorithms by subsequent sensors registrations.

2.2 Forward Dispersion Model

In Bayesian approach the problem of finding the most probable localization of contamination source is reduced to the problem of specification of set of dispersion model's parameters, which depends on the applied model. The aim is to find the source distribution for which model will generate concentrations closest to those actually measured. To do so, a forward dispersion model is needed to calculate the concentration C_i^M at the points i of sensors locations for the tested set of model parameters M at each algorithm step. The applied model can not be to complicated as far the time of computation is crucial. Moreover, advanced dispersion models require large set of parameters to be specify, which leads to the large space of scanned parameters. In this paper we have adopted the fast-running Gaussian plume dispersion model (eg. [6]) as the forward model. This choice was motivated by the requirement of the limiting the computational time and the requirement to do not use the same model like was used to generate the testing synthetic data.

The Gaussian plume dispersion model for uniform steady wind conditions can be written as follows:

$$C(x,y,z) = \frac{q}{2\pi\sigma_y\sigma_z\bar{V}} exp\left[-\frac{1}{2}\left(\frac{y}{\sigma_y}\right)^2\right] \times \\ \{exp\left[-\frac{1}{2}\left(\frac{z-H}{\sigma_z}\right)^2\right] + exp\left[-\frac{1}{2}\left(\frac{z+H}{\sigma_z}\right)^2\right]\} \quad (1)$$

where $C(x,y,z)$ is the concentration at a particular location, V is the wind speed directed along x axis, q is the emission rate or the source strength and H is the height of the release; y and z are the distance along horizontal and vertical direction, respectively; σ_y and σ_z are the standard deviation of concentration distribution in the crosswind and vertical direction. We assume to work under Pasquill stability type C in the urban environment for which the Briggs formula are [7]: $\sigma_y = 0.22x \cdot (1 + x \cdot 4 \cdot 10^{-5})^{-0.5}$ and $\sigma_z = 0.2x$. However, in scanning algorithm we assume that we do not know exact behavior of the plume and consider those coefficients as not completely known. Thus, they are taken as $\sigma_y = \zeta_1 \cdot x \cdot (1 + x \cdot 4 \cdot 10^{-5})^{-0.5}$ and $\sigma_z = \zeta_2 \cdot x$ where values ζ_1 and ζ_2 are sampled by algorithm within interval $[0, 0.4]$.

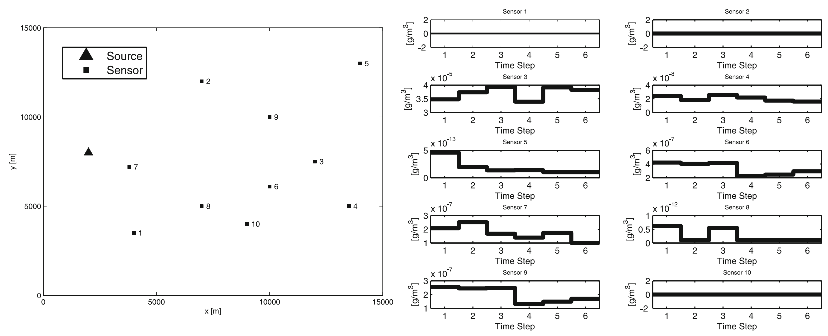


Fig. 1. Distribution of the sensors and the release's source over the domain(left panel) and the synthetic concentration registered by the 10 sensor in 6 subsequent intervals (right panel)

To summarize, in this paper the searched model's parameters' space is:

$$M = (x, y, q, \zeta_1, \zeta_2). \tag{2}$$

3 Theoretical Backgrounds

3.1 Bayesian Inference

A good introduction to Bayesian theory can be found in [8]. Bayes' theorem, as applied to an emergency release problem, can be stated as follows:

$$P(M|D) \propto P(D|M)P(M) \tag{3}$$

where M represents possible model configurations or parameters and D are observed data. For our problem, Bayes' theorem describes the conditional probability $P(M|D)$ of certain source parameters (model configuration M) given observed measurements of concentration at sensor locations (D). The probability $P(D|M)$, for fixed D, is called the likelihood function, while $P(M)$ is the prior distribution. To estimate the unknown source parameters M using Eq. (3), the posterior distribution $P(M|D)$ must be sampled.

Value of likelihood function for a sample is computed by running a forward dispersion model with the given source parameters M and comparing the model predicted concentrations in the points of sensors location with actual data D, as:

$$ln[P(D|M)] = ln[\lambda(M)] = -\frac{\sum_{i=1}^{N}[log(C_i^M) - log(C_i^E)]^2}{2\sigma_{rel}^2} \tag{4}$$

where C_i^M are the predicted by the forward model concentrations at the sensor locations i, C_i^E are the sensor measurements, N is the number of sensors; and σ_{rel}^2 is the standard deviation of the combined forward model and measurement errors. The value of σ^2_{rel} can be varied depending on the observation errors and model formulation. The closer the predicted values are to the measured ones, the higher is the likelihood of the sampled source parameters. After calculation value of the likelihood function for the proposed state its acceptance is performed as: $\frac{ln(\lambda_{prop})}{ln(\lambda)} \geq U(0,1)$ where λ_{prop} of the likelihood value of the proposal state, λ is the previous likelihood value, and $U(0,1)$ is a random number generated from a uniform distribution in the interval $(0,1)$. This condition is more likely to be satisfied if the likelihood of the proposal is only slightly lower than the previous likelihood value. It gives a chance to choose even a little "worse" state, because the probability of acceptance depends directly on the quality of proposed state.

We use a sampling procedure with the Metropolis-Hastings algorithm to obtain the posterior distribution $P(M|D)$ of the source term parameters given the concentration measurements at sensor locations [8,9]. This way we completely replace the Bayesian formulation with a stochastic sampling procedure to explore the model parameters' space and to obtain a probability distribution for the source location.

The posterior probability distribution (3) is computed directly from the resulting samples defined by the algorithm described below and is estimated with

$$P(M|D) \equiv \hat{\pi}^N(M) = \frac{1}{N}\sum_{i=1}^{N}\delta(M_i - M) \tag{5}$$

which represents the probability of a particular model configuration M giving results that match the observations at sensors locations. Equation (5) is a sum over the entire samples set of length N of all the sampled values M_i. Thus $\delta(M_i - M) = 1$ when $M_i = M$ and 0 otherwise. In the case of MCMC parts interpretation is as follows: if a Markov chain spends several iterations at the same location value of $P(M|D)$ increases through the summation.

3.2 Sequential Monte Carlo

SMC is designed to sample from dynamic posterior distributions, both in terms of use the dynamic nature of the model and also in terms of reusing previous calculations eg. [10,11].

Sequential Importance Resampling. Sequential importance resampling (SIR) is a sequential version of importance sampling (IS) and combines IS with resampling procedure [12] . At the center of the SMC approach in our case is the generation of a weighted sample using IS method. IS uses a proposal distribution $q(.)$, that is close to target distribution $\pi(.)$ and from which it is easy to generate samples. The basic methodology is given below:

1. Generate a sample of size N from the proposal distribution $q(M)$:
   $M_{(i)} \sim q(M), i = 1, ..., N.$

2. Compute the importance weights: $\breve{w}(M_{(i)}) \propto \frac{\pi(M_{(i)})}{q(M_{(i)})}, i = 1, ..., N$ and define $w(M_{(i)}) = \frac{\breve{w}(M_{(i)})}{\sum_{j=1}^{N} \breve{w}(M_{(j)})}$.
3. The distribution $\pi(\cdot)$ is then approximated by $\breve{\pi}^N(M) \equiv \sum_{i=1}^{N} w(M_{(i)})\delta(M_i - M)$ which places the probability mass $w(M_{(1)}), ..., w(M_{(N)})$ on the support points $M_{(1)}, ..., M_{(N)}$.

Hence, the weights would be proportional to the value of likelihood (4). In our case to calculate the weight we use of the following formula:

$$\breve{w}(M_{(i)}) \propto -\frac{1}{ln[\lambda(M_{(i)})]}, i = 1, ..., N \tag{6}$$

Resampling is used to avoid the situation when almost all (except only a few) of the importance weights are close to zero. Resampling procedure create new set of samples based on the existing particles (particle = sample + weight). It is similar to repeating sampling with replacement. Basic idea of resampling methods is to eliminate samples which have small normalized importance weights and to concentrate upon samples with large weights. So, for $i = 1, ..., N$ are chosen samples with an indexes $k(i)$ distributed according to the discrete distribution with N elements satisfying $P(k(i) = l) = w(M_{(i)})$ for $l = 1, ..., N$; then for $i = 1, ..., N$ for samples $M_{k(i)}$ are assigned the weights $w(M_{(i)}) = \frac{1}{N}$.

Effective sample size is $\hat{N}_{eff} = \frac{1}{\sum_{i=1}^{N} w(M_{(i)})^2}$ where $w(M_{(i)})$ are normalized weights. If all weights are equal $1/N$ then effective sample size is N. In the contrast to a situation where all weights = 0, except for one weight = 1, effective sample size is equal 1.

Sequential Monte Carlo Algorithms. The SMC algorithm needs some set of samples to be initialized. An ideal way to generate this initial sample is use MCMC data from first K iterations in all time steps. The resulting equally weighted MCMC set of samples can then be passed on to SMC for processing in the subsequent iteration.

We assume that the information from the sensors arrive subsequently in six time steps. We start to search for the source location (x, y), release rate (q) and model parameters ζ_1 and ζ_2 after first sensors' measurements (based on the data in time $t = 1$, see (Fig. 1)). Thus, scanning algorithm is run with obtaining the first measurements from the sensors. Based on this information we obtain the probability distributions of the searched parameters (5) starting from the randomly chosen set of parameters M. The forward calculation are performed for the actual state M and likelihood function λ is calculated. Then we apply random walk procedure "moving" our Markov chain to the new position. Details of the algorithm can be found in [3,4]. After K iteration we pass all the samples (from all 10 chains)to the sequential procedure. We compute importance weights by and normalize them, next we use roulette procedure to draw $\hat{N}_{eff}$ samples from the set generated by Markov Chain.

In this paper, we consider the following variants of scanning algorithms:

1. **MCMC.** In this algorithm, the parameter space scan in each time step t is independent form the previous ones. So, in this case we don't use information from past calculations. MCMC don't use sequential mechanism.
2. **SMC via Maximal Weights.** In subsequent SMC calculations algorithms uses the results obtained by SMC in the previous time steps to run calculation with use of the new measurements. As the first location of Markov chain M_0^t it select the set of M parameters for which weight in previous time step was the highest. So, for $t > 1$:
$M_0^t \sim \ arg \ (M \in \{M_0^{t-1}, ..., M_n^{t-1}\}) \ max \ w(M_i^{t-1})$. With this approach, we always start with the best values of the model (previously found) and correct the result with new information from sensor.
3. **SMC via Rejuvenation and Extension.** In contrast to the SMC via Maximal Weights this algorithm as the first location of Markov chain M_0^t at the time $t > 1$ chooses the set of parameters M selected randomly from previous realization of resampling procedure in $t-1$ with use of the uniform distribution: $M_0^t \sim U(M_0^{t-1}, M_1^{t-1}, ..., M_n^{t-1}) \ a \ uniform \ distribution \ \{1, ..., n\}$. Applying the new knowledge (new measurements) the current chain is "extended" starting from selected position with use of the new data in the likelihood function calculation.

4 Results

Algorithms described in Sect. 3.2 have been tested on the same synthetic data set (Fig. 1). In our calculation we use 10 Markov chains in each time step. Figure 2 presents the traces, of 4 Markov chains and samples after resampling in SMC via Maximal Weight algorithm for 4 time steps, in the location space. The number of iteration n for each Markov chain is $n = 8000$. This number was chosen

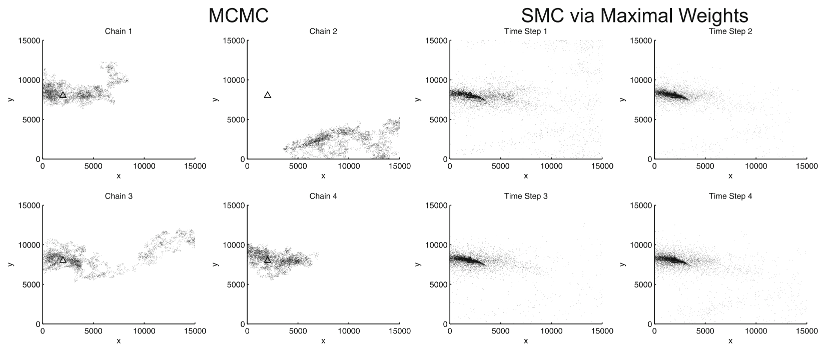


Fig. 2. The traces of four Markov chains in the location space for MCMC (left panel) and samples after resampling in SMC via Maximal Weight algorithm for 4 time steps. The source location is marked by triangle

based on the numerical experiments as the number of iteration needed to reach convergence for each sampled model parameters M. Statistical convergence to the posterior distribution was monitored by computing between-chain variance and within-chain variance [3,8]. One of the important aspects of stochastic procedure of calculating the posterior distribution is choosing burn-in phase. The burn-in factor represents the number of samples needed at the beginning for the Markov chain to actually reach the state when it is sampling from the target distribution. These initial samples are discarded and not used for inference. In our calculation the burn-in was fixed at 1000 iterations.

Figures 3 and 4 shows the marginal probability distribution for x and y coordinates of source location within the considered domain calculated by MCMC and SMC via Maximal Weights algorithms. The exact source location, set up

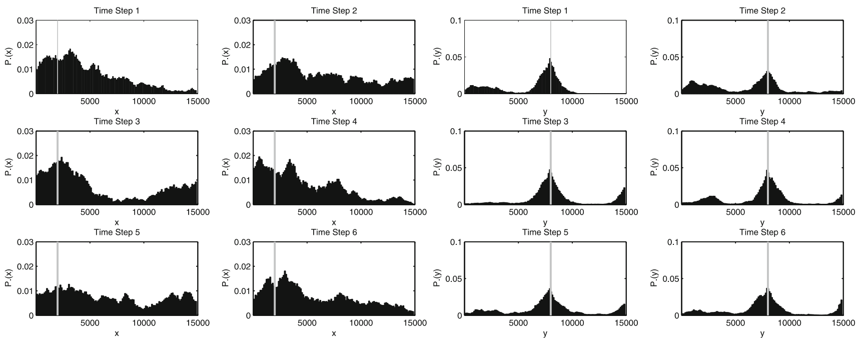


Fig. 3. Posterior distribution as inferred by the Bayesian event reconstruction for MCMC algorithm for x an y parameters. Posterior distributions were averaged based on the data for all Markov chains in each time step. Vertical lines represent the target value.

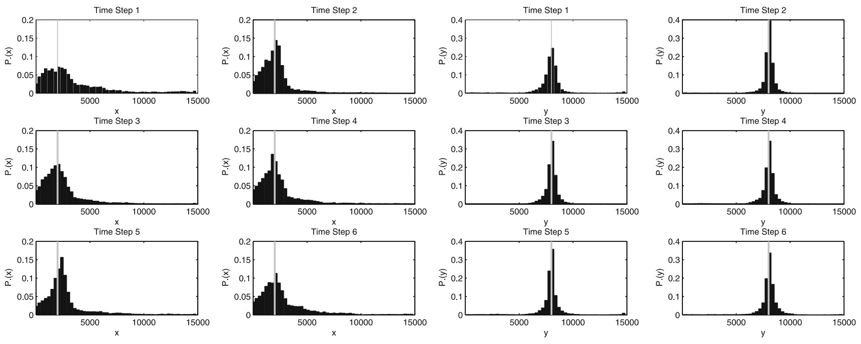


Fig. 4. Posterior distribution as inferred by the Bayesian event reconstruction for SMC via Maximal Weights algorithm for x an y parameters. Posterior distributions were averaged based on the data for all samples in each time step. Vertical lines represent the target value.

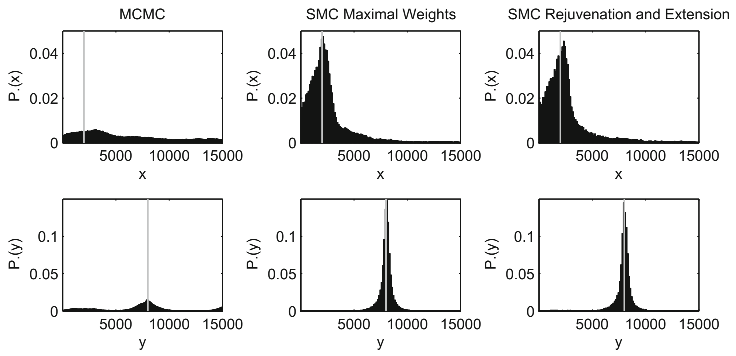


Fig. 5. Posterior distribution as inferred by the Bayesian event reconstruction for all applied algorithms for x an y parameter. Posterior distributions were averaged over data for all time steps. Vertical lines represent the target value.

in creation of the testing data, is marked by the vertical line. The distributions obtained from SMC via Rejuvenation and Extension were very similar to results of SMC via Maximal Weights. One can see that all algorithms found, with high probability, the contamination source location in the crosswind y direction. The high peek in the histogram is justified by the sensitivity of the used forward dispersion model to this parameter. Another situation is with the x coordinate of the source. The MCMC algorithm do not find the right x value. At the same time for the two SMC algorithms the synthetic true answer lies within a region of high posterior probability. It is worth to mention that these algorithms (SMC via Maximal Weights, SMC via Rejuvenation and Extension) use the probability distributions obtained based on information from previous measurements to update the distribution with use of the new data which can be seen from Fig. 4. This methodology makes these algorithms more effective in location the most probable value of considered parameters. However, none of the methods found the correct release rate q. This is caused by the simplicity of the applied forward dispersion model. We do not consider here the results for ζ_1, ζ_2 due to difference of the SCIPUFF and Gaussian models in its estimation.

To effectively compare the results given by all proposed algorithms we have estimated the joint marginal distribution of x and y parameters. Figure 5 presents the posterior distributions averaged based on the data for all time steps for all samples. We see that, in contrary to MCMC method, both modifications of SMC methods successfully find the true location of contaminant source assumed in creation of the synthetic data.

5 Conclusions

We have presented a methodology to reconstruct a source of contamination based on a set of sparse measurements. The method combines Bayesian inference with SMC sampling and produces posterior probability distributions of the parameters describing the unknown source. Developed dynamic data-driven event reconstruction model, which couples data and predictive models through Bayesian inference, successfully found the solution to the stated inverse problem i.e. having the downwind concentration measurement and knowledge of the wind field algorithm found the most probable location of the source. We have examined various version of the SMC algorithms i.e. SMC via Maximal Weights, SMC via Rejuvenation and Extension and compare its effectiveness to estimate the probabilistic distributions of searched parameters with MCMC. We have shown the advantage of the SMC algorithms that in different ways use the source location parameters probability distributions obtained basing on available measurements to update the marginal probability distribution. The probabilistic aspect of the solution optimally combines a likely answer with the uncertainties of the available data. Among several possible solutions, the Bayesian source reconstruction is solely able to find values of the model parameters that are more consistent with the data available and its uncertainties. The stochastic approach used in this paper is completely general and can be used in other fields where the parameters of the model bet fitted to the observable data should be found.

Acknowledgments. This work was supported by the Welcome Programme of the Foundation for Polish Science operated within the European Union Innovative Economy Operational Programme 2007-2013 and by the EU and MSHE grant nr POIG.02.03.00-00-013/09. The work was supported by the project VI.B.08 under the NCBiR national programme: “Improving labour and safety conditions: 2nd stage”.

References

1. Watzenig, D.: Bayesian inference for inverse problems - statistical inversion. Elektrotech. Informationstechnik **124**(7–8), 240–247 (2007)
2. Senocak, I., Hengartner, N.W., Short, M.B., Daniel, W.B.: Stochastic event reconstruction of atmospheric contaminant dispersion using Bayesian inference. Atmos. Environ. **42**(33), 7718–7727 (2008)
3. Borysiewicz, M., Wawrzynczak, A., Kopka, P.: Stochastic algorithm for estimation of the model’s unknown parameters via Bayesian inference. In: Proceedings of the Federated Conference on Computer Science and Information Systems, pp. 501–508. IEEE Press (2012)
4. Borysiewicz, M., Wawrzynczak, A., Kopka, P.: Bayesian-based methods for the estimation of the unknown model’s parameters in the case of the localization of the atmospheric contamination source. Found. Comput. Decis. Sci. **37**(4), 253–270 (2012)
5. Sykes, R.I. et al.: PC-SCIPUFF Version 1.2PD Technical Documentation. ARAP Report No. 718. Titan Corporation (1998)
6. Turner, D.B.: Workbook of Atmospheric Dispersion Estimates. Lewis Publishers, USA (1994)
7. Panofsky, H.A., Dutton, J.A.: Atmospheric Turbulence. John Wiley, New York (1984)
8. Gelman, A., Carlin, J., Stern, H., Rubin, D.: Bayesian Data Analysis. Chapman & Hall/CRC, Boca Raton (2003)

9. Gilks, W., Richardson, S., Spiegelhalter, D.: Markov Chain Monte Carlo in Practice. Chapman & Hall/CRC, Boca Raton (1996)
10. Doucet, A., de Freitas, J.F.G., Gordon, N.J.: Sequential Monte Carlo Methods in Practice. Springer, New York (2001)
11. Liu, J.S.: Monte Carlo Strategies in Scientific Computing. Springer, New York (2001)
12. Gordon, N.J., Salmond, D.J., Smith, A.F.M.: Novel approach to nonlinear/non-Gaussian Bayesian state estimation. IEEE Proc. Radar Signal Process. **140**(2), 107–113 (1993)

Effective Parallelization of Quantum Simulations: Nanomagnetic Molecular Rings

Piotr Kozłowski[1], Grzegorz Musiał[1](✉), Michał Antkowiak[1], and Dante Gatteschi[2]

[1] Faculty of Physics, Adam Mickiewicz University, ul. Umultowska 85, PL-61-614 Poznań, Poland
gmusial@amu.edu.pl

[2] Department of Chemistry, INSTM and Universitá di Firenze, via della Lastruccia 3, I-50019 Sesto Fiorentino, Italy

Abstract. The effective parallelization of processing exploiting the MPI library for the numerically exact quantum transfer matrix (QTM) and exact diagonalization (ED) deterministic simulations of chromium-based rings is proposed. In the QTM technique we have exploited parallelization of summation in the partition function. The efficiency of the QTM calculations is above 80 % up to about 1000 processes. With our test programs we calculated low temperature torque, specific heat and entropy for the chromium ring Cr_8 exploiting realistic Hamiltonian with single-ion anisotropy and the alternation of the nearest neighbor exchange couplings. Our parallelized ED technique makes use of the self-scheduling scheme and the longest processing time algorithm to distribute and diagonalize separate blocks of a Hamiltonian matrix by slave processes. Its parallel processing scales very well, with efficiency above 90 % up to about 10 processes only. This scheme is improved by processing more input data sets in one job which leads to very good scalability up to arbitrary number of processes. The scaling is improved for both techniques when larger systems are considered.

Keywords: Parallelization of processing · MPI · Numerical simulations · Nanomagnetic rings · Heisenberg model

1 Introduction

The parallelization of processing based on the MPI library [1] for modeling of molecular nanomagnets is proposed. Message passing model is exploited, as it effectively works in every computer system with the distributed, shared or mixed type of memory and fits well separate processors connected by fast or slow communication network. Molecular rings made of transition-metal ions immersed in a non-magnetic environment became of recent interest as potential building blocks of envisaged hybrid or quantum computers or as a basis of an efficient storage device [2]. The intra-cluster ferro- or antiferromagnetic interactions are usually modeled with a Heisenberg Hamiltonian. Yet, there are subtle, but important for future applications, phenomena like e.g. S-mixing, which cannot be described within the giant spin approximation [3] and require inclusion of anisotropic terms to the Heisenberg Hamiltonian [4,5]. Anisotropy lowers the symmetry of the model and thus makes exact algebraic approaches infeasible due to increase of computational complexity of the problem. Therefore for anisotropic models the perturbative methods are usually used instead.

To avoid the problems of accuracy and limited range of application pertinent to many perturbative methods we have developed [6–8] parallel versions of two exact computational techniques: quantum transfer matrix (QTM) and exact diagonalization (ED), using MPI [1]. In this paper we test the scalability of both methods applied to simulations of the ring shaped molecular nanomagnet Cr_8 (QTM and ED) and its derivative $\mathrm{Cr}_8\mathrm{Ni}$ (ED) on several high performance multicomputers.

Our work supplements some efforts to include parallel processing in magnetism (see e.g. openMP parallelization for Householder [9] and Lanczos [10] methods).

2 Physical Setup

The Cr_8 ring can be modeled using the quantum spin ($s = 3/2$) Hamiltonian of the form

$$\begin{aligned}\mathcal{H} = &\sum_{j=1}^{4} \left(J^{\mathrm{o}} \boldsymbol{s}_{2j-1} \cdot \boldsymbol{s}_{2j} + J^{\mathrm{e}} \boldsymbol{s}_{2j} \cdot \boldsymbol{s}_{2j+1}\right) \\ &+ \sum_{j=1}^{8} \left[D (s_j^z)^2 - g\mu_{\mathrm{B}} B \left(s_j^x \sin\theta + s_j^z \cos\theta\right)\right] ,\end{aligned} \tag{1}$$

where $J^{\mathrm{o,e}}$ are the nearest-neighbor exchange integrals for ‘odd’ and ‘even’ pairs, respectively, D is the site-independent single-ion anisotropy, B is the external magnetic field applied in the x-z plane, whose direction forms an angle θ with the z axis, g is the corresponding Landé factor and μ_{B} stands for Bohr magneton. We assume periodic boundary conditions ($j + 8 \equiv j$) because of the ring geometry of the molecule. The physical quantities of interest: the magnetic torque τ, the specific heat C and the entropy S are defined as follows:

$$\tau^y = -g\mu_{\mathrm{B}} B \left(\langle S^x \rangle \cos\theta - \langle S^z \rangle \sin\theta\right), \; C = -T \left(\frac{\partial^2 F}{\partial T^2}\right)_B, \; S = -\left(\frac{\partial F}{\partial T}\right)_B \tag{2}$$

We specify here the only non-zero y-th component of the magnetic torque as the magnetic field is in the x-z plane. The free energy F and the thermal averages of the total spin components $\langle S^x \rangle$ and $\langle S^z \rangle$ present in Eq. (2) can be calculated from the partition functions Z, Z^x and Z^z:

$$F = -k_{\mathrm{B}} T \ln Z \; , \quad \langle S^{x,z} \rangle = \frac{Z^{x,z}}{Z} \; , \quad Z = \mathrm{Tr} \mathrm{e}^{-\beta\mathcal{H}} \; , \quad Z^{x,z} = \mathrm{Tr} S^{x,z} \mathrm{e}^{-\beta\mathcal{H}} \; . \tag{3}$$

The traces in Eq. (3) go over 4^8 dimensional spin space and are hard to compute because of their size and the presence of non-commuting operators in $\mathcal{H}$.

To overcome these problems we employ the transfer matrix technique to factorize $\exp(-\beta\mathcal{H})$ and to obtain the results in reasonable time, we run our program in many parallel processes. For the magnetic field along the zth axis ($\theta = 0$) we have also performed exact diagonalization of the Hamiltonian (1), calculating both eigenvalues and eigenvectors.

3 Quantum Transfer Matrix Technique and Parallel Processing

Hamiltonian (1) can be written as a sum of two non-commuting Hamiltonians $\mathcal{H}^{\rm o}$ and $\mathcal{H}^{\rm e}$. Then, with the help of the Trotter formula, the related partition function Z can be expressed as the limit:

$$Z = \lim_{m\to\infty} Z_m = \lim_{m\to\infty} \mathrm{Tr}\left(\mathrm{e}^{-\frac{\beta}{m}\mathcal{H}^{\rm o}}\mathrm{e}^{-\frac{\beta}{m}\mathcal{H}^{\rm e}}\right)^m . \tag{4}$$

To optimize the numerical calculations the m-th approximant to the partition function (Z_m) is represented as a product of the matrices V^o, V^e and shift operators $\mathcal{P}$, $\mathcal{P}^\dagger$ ([6–8] and references therein)

$$Z_m = \mathrm{Tr}\left[\left(V^{\rm o}\mathcal{P}^\dagger\mathcal{P}^\dagger\right)^4\mathcal{P}^\dagger\left(V^{\rm e}\mathcal{P}^\dagger\mathcal{P}^\dagger\right)^4\mathcal{P}\right]^m , \tag{5}$$

with $V^{\rm o,e} = \mathrm{e}^{H_{12}^{o,e}} \otimes \hat{1} \otimes \hat{1} \otimes \hat{1}$[1], being sparse matrices with no more than 16 non-zero elements in each column and each row. The two-spin Hamiltonians $H_{12}^{o,e}$ are defined as follows:

$$\begin{aligned} H_{12}^{o,e} = J^{o,e}\left(s_1^z s_2^z + \frac{1}{2}(s_1^+ s_2^- + s_1^- s_2^+)\right) + \frac{1}{2}\left(D[(s_1^z)^2 + (s_2^z)^2]\right. \\ \left. - g\,\mu_{\rm B}B\left[(s_1^z + s_2^z)\cos\theta + (s_1^x + s_2^x)\sin\theta\right]\right) . \end{aligned} \tag{6}$$

Only these two Hamiltonians have to be diagonalised to calculate matrices $V^{o,e}$ and then Z_m. One can calculate approximants Z_m^x and Z_m^z in a similar way.

The trace in (5) is calculated by acting with the operators $V^{\rm o,e}$, $\mathcal{P}$ and $\mathcal{P}^\dagger$ on all the base vectors. However, we exploit the symmetry of the system and take into account only non-equivalent vectors reducing in this way the number of operations approximately by a factor of 1/8. Due to the sparsity of $V^{\rm o,e}$, $\mathcal{P}$ and $\mathcal{P}^\dagger$ only $m4^8$ operations have to be performed in (5) instead of $2m4^{16}$ for regular matrix vector multiplication.

The thermodynamic quantities are obtained by extrapolation of the corresponding approximants (to $m \to \infty$) which for large enough m should be linear functions of $1/m^2$ [11]. Unfortunately, for our system at low temperatures these linear extrapolations are justified only for very big values of m ($\sim 10^3 \div 10^4$) which is a challenge for scalability of our parallelized programs.

[1] The symbol $\otimes$ denotes the tensor product and $\hat{1}$ is a 16×16 identity operator.

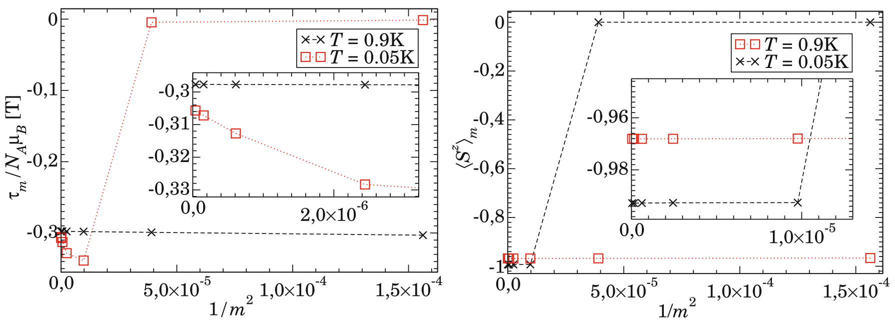


Fig. 1. The m dependence of the torque τ^y_m and the z-th component of total spin $\langle S^z\rangle_m$ approximants for Cr$_8$ in the field $B = 10\,T$ for two different temperatures. $J^{\rm o} = J^{\rm e} = 16,94\,K$, $D = -0,34\,K$, $\theta = 6°$

One can see in the insets of Figs. 1 and 2 that the linearity of various approximants (τ^y_m, $\langle S^z\rangle_m$, and C_m) as functions of $1/m^2$ at temperature $0.05\,K$ becomes apparent only for $m > 1000$. We notice that at $T = 0.05\,K$ dependence of τ_m and $\langle S^z\rangle_m$ on $1/m^2$ is approximately linear also for smaller values of m $(40 < m < 160)$, but the appropriate extrapolations give wrong results. For the entropy at temperature $T = 0.05\,K$ (Fig. 2) one has to go even beyond $m = 5000$ to obtain physically correct results i.e. $S > 0$. For such large values of m $(m > 1000)$ the execution times of the sequential job are equal to a couple of days even if relatively small systems like Cr$_8$ are considered.

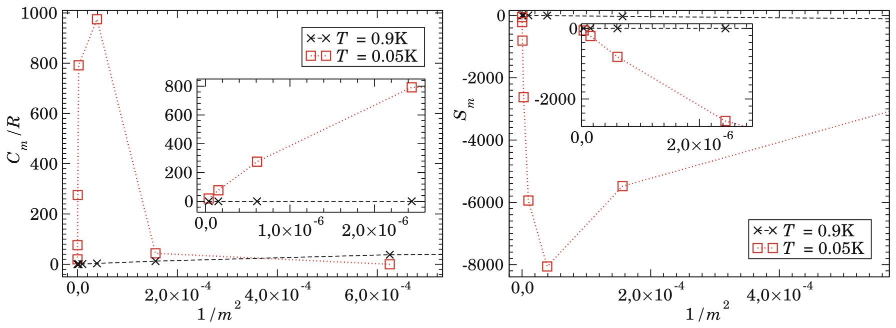


Fig. 2. The m dependence of the specific heat C_m and entropy S_m approximants for Cr$_8$ in the field $B = 7\,T$ for two different temperatures. $J^{\rm o} = J^{\rm e} = 16,94\,K$, $D = -0,34\,K$, $\theta = 65°$

These results show the large scale of challenges in modeling of nanomagnets and motivated us to parallelize our Fortran code for distributed computer systems with message passing implemented in the MPI library. Summation in Eq. (5) is distributed uniformly among parallel processes to calculate the partition functions Z_m, Z^x_m and Z^z_m. We have used function MPI_Reduce which organizes tree-like communication.

4 Exact Diagonalization Technique and Parallel Processing

The exact diagonalization technique is also a powerful tool to calculate the thermodynamic quantities. However, for the nanomagnet under consideration the size of the Hamiltonian matrix ($4^8 \times 4^8$), makes sequential processing of numerical diagonalization impossible in realistic time if no additional conditions are imposed. The ED technique is used for $\theta = 0$ in Eq. (1). In this case the Hamiltonian takes on a block diagonal form, in the basis formed by eigenvectors of the z projection of the total spin, i.e. $S^z = \sum_{j=1}^{8} s_j^z$. The 48 diagonal matrix blocks can be unequivocally labeled by the eigenvalues M of S^z and by the symmetry a of the eigenstate, with $a = 1$ for symmetric vectors and $a = 0$ for antisymmetric ones.

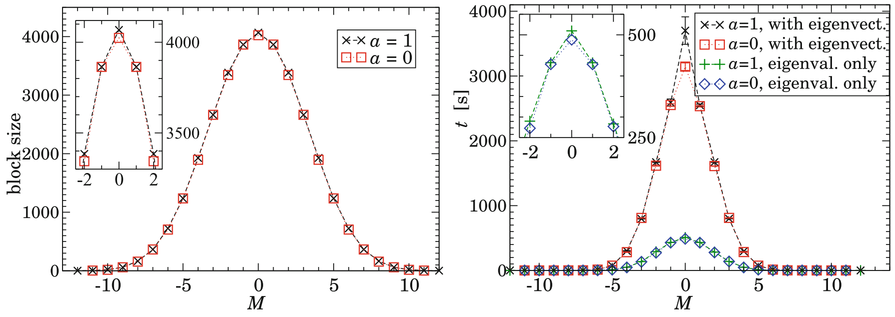


Fig. 3. Left: Sizes of the Hamiltonian matrix blocks for different values of M. Right: The processing times t for individual Hamiltonian matrix blocks: two lower curves illustrate the calculation time of the eigenvalues only, whereas two upper ones stand for the calculation time of both the eigenvalues and the eigenvectors

The main difficulty in parallel ED processing is due to the sizes of matrix blocks which differ from one another by several orders of magnitude varying from 4068×4068 to 1×1 for Cr_8 nanomolecule, as shown in the left graph of Fig. 3. For even M's the size of symmetric boxes ($a = 1$) is always greater than that of antisymmetric ones ($a = 0$), whereas for odd M's these sizes are equal to each other. For the lowest M only symmetric 1×1 boxes exist.

The right graph of Fig. 3 presents processing times for calculation of eigenvalues only and eigenvalues together with eigenvectors for all the matrix blocks. Thus, for the Cr_8 nanomagnet the job has to perform 48 independent diagonalization tasks for each Hamiltonian matrix block which can be processed concurrently.

To parallelize our C++ code for distributed computer systems, we have used message passing implemented in the MPI library in the master-slave model of communication with the asymmetric star-like topology. It means that the processes responsible for diagonalization of the largest blocks communicate most rarely with the master process. The master process fulfills only the scheduling role and assigns tasks to p slave processes, using Longest Processing Time (LPT) algorithm: the tasks are sorted by the block size and are assigned in decreasing order [12]. The self-scheduling scheme organizing the work of $p+1$ parallel processes in our program uses the MPI_ANY_SOURCE and MPI_ANY_TAG arguments in the function MPI_RECV call, so that the master process always first receives a result which was calculated most recently. Each of p slave process calculates the block of the Hamiltonian matrix labeled by the numbers M and a obtained from the master process, diagonalizes it and sends the eigenvalues back to the master process. When any of the slave processes finishes its job, the master process receives calculated eigenvalues and sends back the label of the next block waiting in a sorted queue or MPI_BOTTOM when the queue is already empty. After obtaining MPI_BOTTOM a slave stops.

5 Quantum Transfer Matrix Results

Our QTM simulations described above were first performed on multicomputer "reef" built up of 22 nodes with two dual-core Intel Xeon 3 GHz CPUs and of 122 nodes with two quad-core Intel Xeon 2.33 GHz CPUs with OpenMPI library [1] and InfiniBand technology of interconnect links. The main part of processing within our program consists in parallel calculation of the partition functions (Eq. (5)) distributed uniformly among parallel processes as explained above.

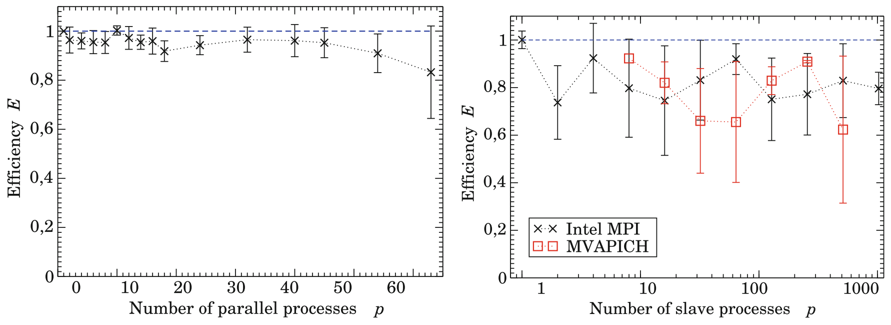


Fig. 4. The dependence of the efficiency E on the number of parallel processes p in the QTM simulations run on "reef" (left) and "galera" (right). The ideal efficiency is shown with a broken line. The error bars mark a standard deviation

To determine scaling of our simulations with p parallel processes, we calculate efficiency E defined as $u = t_{seq}/(pt_{par})$ [13], where t_{seq} and t_{par} denote the sequential and parallel execution times of our program, respectively. The statistics for each value p was estimated from several runs, varying the number of cores per node used.

The p dependence of efficiency E of our simulations performed on multicomputer "reef" presented in the left graph of Fig. 4 proves good, though not perfect, scalability of processing in our program. No significant differences in execution times were observed if various number of nodes for a fixed number of CPU cores were engaged. The efficiency seems to have two small maxima for $p = 10$ and $p = 32$ what may suggest the optimal number of processes for practical calculations.

The sequential execution of our test program for Cr_8 nanomolecule takes about 52 h. The communication between the parallel processes takes rather small fraction of the execution time since the slave processes send to the master process only few numbers at the end of the execution. Therefore, most of the error bars in the left graph of Fig. 4, also for a big number of processes, touch the value 1 of efficiency E. It means that for almost each number p processing within our program can scale perfectly (cf. [14]).

The additional tests have been made on a larger multicomputer "galera" consisting of 672 nodes, each with two Intel Xeon Quad Core 2,33 GHz CPUs and InfiniBand technology of interconnect links. We tested two different MPI implementations: Intel MPI and MVAPICH. The results for efficiency E are presented in the right graph in Fig. 4 in log-linear scale. Here the sequential execution of our programs takes only about 13 h. The statistics for each number p of parallel processes is done from several runs. For all values of p large oscillations (up to 50 %) of the execution time, expressed by the error bars (see the right graph in Fig. 4), can be observed. It should be pointed out, that for almost each p the shortest runs scale perfectly ($E = 1$) which means that the managing system of the multicomputer should be blamed for lower efficiency. Less optimal scaling observed for $p = 1024$ is additionally due to the size of our model. The sum in Eq. (5) for Cr_8 goes over 8355 non-equivalent states which cannot be uniformly distributed among $p = 1024$ parallel processes so that each of them has to process only 8 or 9 elements of the sum.

Our parallelized program described above enabled us to calculate the torque τ for very low temperatures (below $1\,K$), hardly accessible with sequential processing. At $T = 50\,mK$ we had to take the large (up to 5120) values of m (see Eq. 4). The specific heat C and the entropy S were calculated in slightly higher temperatures (around 1 K) to compare them to the experimental data. The more detailed description and discussion of the physical results can be found in [15].

6 Exact Diagonalization Results

The same multicomputer "reef" has been used for parallel simulations of Cr_8 nanomolecular ring by means of the ED technique described above. High efficiency ($E > 0.9$) demonstrating good scalability of our problem with respect to the number of slave processes p can be observed (the left graph of Fig. 5) only up to some value $p = p_{max}$, with $p_{max} = 10$ when only eigenvalues are calculated and $p_{max} = 9$ when both eigenvalues and eigenvectors are determined. This limit in scalability follows from the above explained structure of the problem: the sizes of matrix blocks vary from 4068×4068 to 1×1. Thus, only a few matrix blocks of the Hamiltonian are of large size (see Fig. 3). For larger values of p the

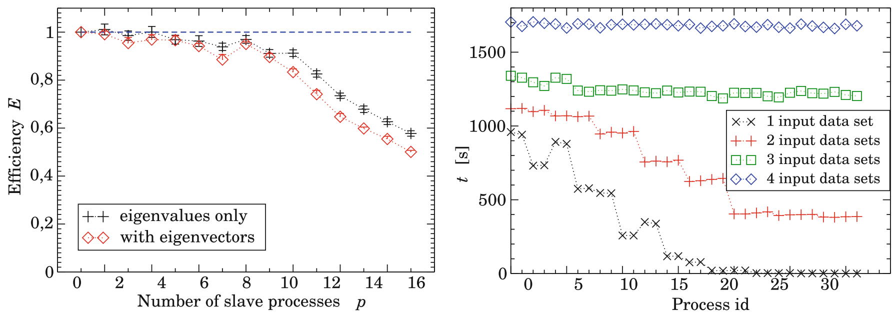


Fig. 5. Left: The dependence of efficiency E on the number of parallel processes p for the ED method. The ideal efficiency is shown with a broken line, the error bars mark standard deviation. Right: The execution times as a function of process id for one job distributed over 32 parallel processes with 1, 2, 3 and 4 input data sets.

execution time is dominated by the time spent on diagonalization of the largest blocks, which cannot be reduced. Adding more slave processes leads to poorer efficiency as the total execution time cannot be shorter than time needed for diagonalization of the largest matrix block corresponding to $M = 0$. Therefore for $p > p_{max}$, for which the total execution time is equal to the diagonalization time of the largest blocks, one can see decrease of the efficiency in the left graph of Fig. 5. Also here error bars for each p are estimated on the basis of several runs with variation of the number of cores per node used.

For larger molecules, e.g. larger chromium-based rings, sequential processing time dramatically increases and one can hardly obtain results in realistic time. However in this case also the number of large diagonal blocks in the Hamiltonian increases, which should gives rise to better scalability (increase of p_{max}).

In order to extend the limited scalability of our ED simulations, we have added the possibility to perform concurrent processing for many initial data within one job. The same LPT algorithm was used but it has got more matrix blocks available. This leads to lower latencies in parallel processes. The simulations have been performed on multicomputer "huygens" with 3328 Power6 4.7 GHz processor cores in SARA Computing and Networking Services in Amsterdam. The results of our tests for the larger Cr_8Ni nanomolecule performed on 32 computing cores are shown in the right graph of Fig. 5.

The large latencies within the job processing one input data set in 32 parallel processes are clearly seen. The reason for these latencies is explained above. The good scalability of the Cr_8Ni nanomolecule simulations is limited to about 12 parallel processes. The right graph in Fig. 5 shows that in the job processing two input data sets concurrently these latencies are much smaller. This graph proves also that processing of three (and more) input data sets concurrently in one job is enough to practically eliminate latencies within 32 parallel processes. For larger number of parallel processes the required number of data sets to be used simultaneously in one job in order to eliminate latencies can be determined in a similar way.

For the data presented in the right graph of Fig. 5 we have used ScaLAPACK procedures for diagonalization of matrix blocks instead of the ones from Numerical Recipes. Thus, the times are shorter than these presented in Fig. 3. Moreover, in this implementation the master process also performs calculations as each of the slave processes.

7 Conclusions

We have analyzed the efficiency of the QTM parallel simulations of the chromium-based molecular ring Cr_8. Our parallelized code enabled us to calculate the torque, the specific heat and the entropy in the low temperature regime, which is inaccessible with sequential processing. These simulations scale very well up to about thousand of parallel processes where the limit due to the size of the system is reached. For larger systems perfect scaling can be extended to much larger number of processes.

We have demonstrated for two chromium rings Cr_8 and Cr_8Ni that up to $p_{max} \sim 10$ processes the simulations based on the ED method and exploiting the LPT algorithm with self-scheduling scheme scale very well giving efficiency $E > 0.9$. The particular value of p_{max} is determined by the number of large diagonal blocks in the Hamiltonian and it increases with the size of the system. It is demonstrated that the extension of the parallelization scheme allowing for simultaneous processing of many input data sets in one job extends scalability to arbitrary number of processes limited only by the number of available input data sets.

It is worth noting that both methods are complementary and can be easily applied to other ring-shaped nanomagnets. QTM provides precise values of thermodynamic quantities for complicated anisotropic Hamiltonians and relatively large systems, whereas ED works better for simpler and smaller systems but gives access to more detailed information in the form of the energy level structure. The QTM technique is usually more time-consuming than ED but has much better scaling properties and requires less computer memory than the latter. In the case of both methods parallelization of processing extends their applicability to much larger and more complex systems.

Acknowledgments. The authors acknowledge helpful discussions with Prof. F. Esposito, M. Kupczyk and M. Pospieszny. This work was supported in part by the MNiSW within the project No. N519 579138. Numerical calculations were carried out on the multicomputer “reef” in the Supercomputing and Networking Center in Poznań, “galera” in the Academic Computer Center in Gdańsk and “huygens” in SARA Computing and Networking Services in Amsterdam. A part of simulations was also performed on multicomputer “pearl” in Faculty of Physics at Adam Mickiewicz University in Poznań (incorporated in a scientific grid environment).

References

1. http://www.mpi-forum.org/ - MPI Forum Home Page
2. Meier, F., Levy, J., Loss, D.: Quantum computing with antiferromagnetic spin clusters. Phys. Rev. B **68**, 134417 (2003)
3. Carretta, S., Santini, P., Affronte, M., Ghirri, A., Sheikin, I., Piligkos, S., Timco, G., Winpenny, R.E.P.: Topology and spin dynamics in magnetic molecules. Phys. Rev. B **72**, 060403(R) (2005)
4. Baker, M.L., Timco, G., Piligkos, S., Mathieson, J.S., Mutka, H., Tuna, F., Kozlowski, P., Antkowiak, M., Guidi, T., Gupta, T., Rath, H., Woolfson, R.J., Kamieniarz, G., Pritchard, R.G., Weihe, H., Cronin, L., Rajaraman, G., Collison, D., McInnes, E.J.L., Winpenny, R.E.P.: A classification of spin frustration in molecular magnets from a physical study of large odd-numbered-metal, odd electron rings. Proc. Natl. Acad. Sci. U.S.A. **109**, 19113 (2012)
5. Antkowiak, M., Kozlowski, P., Kamieniarz, G., Timco, G.A., Tuna, F., Winpenny, R.E.P.: Detection of ground states in frustrated molecular rings by the in-field local magnetization profiles. Phys. Rev. B. **87**, 184430 (2013)
6. D'Auria, A.C., Esposito, U., Esposito, F., Kamieniarz, G., Matysiak, R.: Exact simulations of quantum rings and characterization of hexanuclear manganese and dodecanuclear nickel cyclic complexes. J. Phys. Condens. Matter **13**, 2017 (2001)
7. Kamieniarz, G., Matysiak, R., D'Auria, A.C., Esposito, F., Benelli, C.: Large-scale simulations of the finite-temperature properties of the molecular assemblies Mn6 and Ni12. Comput. Phys. Commun. **147**, 194 (2002)
8. Kamieniarz, G., Matysiak, R.: Simulations of the low-dimensional magnetic systems by the quantum transfer-matrix technique. Comp. Mat. Sci. **28**, 353 (2003)
9. Honecker, A., Schüle, J.: OpenMP implementation of the householder reduction for large complex Hermitian eigenvalue problems. Adv. Parallel Comput. **15**, 271–278 (2008)
10. Schnack, J., Hage, P., Schmidt, H.-J.: Efficient implementation of the lanczos method for magnetic systems. J. Comput. Phys. **227**, 4512 (2008)
11. Suzuki, M.: Quantum statistical monte carlo methods and applications to spin systems. J. Stat. Phys. **43**, 883 (1986)
12. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**, 416 (1969)
13. Van de Velde, E.F.: Concurrent Scientific Computing. Springer, New York (1994)
14. Matysiak, R., Haglauer, M., Kamieniarz, G., D'Auria, A.C., Esposito, F.: Application of parallel computing in the transfer-matrix simulations of the supramolecular rings. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2004. LNCS, vol. 3019, pp. 475–480. Springer, Heidelberg (2004)
15. Kozłowski, P., Musiał, G., Haglauer, M., Florek, W., Antkowiak, M., Esposito, F., Gatteschi, D.: Non-perturbative methods in phenomenological simulations of ring-shape molecular nanomagnets. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2013, Part II. LNCS, vol. 8385, pp. 438–447. Springer, Heidelberg (2014)

DFT Study of the Cr_8 Molecular Magnet Within Chain-Model Approximations

Valerio Bellini[1,2](✉), Daria M. Tomecka[3], Bartosz Brzostowski[4], Michał Wojciechowski[4], Filippo Troiani[1], Franca Manghi[1], and Marco Affronte[1]

[1] CNR-INFM-National Research Center on nanoStructures and bioSystems at Surfaces (S3), Via Campi 213/A, 41100 Modena, Italy
valerio.bellini@unimore.it

[2] Istituto di Struttura della Materia (ISM) - Consiglio Nazionale delle Ricerche (CNR), I-34149 Trieste, Italy

[3] Faculty of Physics, A. Mickiewicz University, ul. Umultowska 85, 61-614 Poznań, Poland

[4] Institute of Physics, University of Zielona Góra, ul. Prof. Szafrana 4a, 65-516 Zielona Góra, Poland

Abstract. We present a density functional theory (DFT) study of the electronic and magnetic properties of the Cr_8 molecular ring. The all-electron linearized augmented plane wave method (LAPW) implemented in the Wien2k package and pseudopotential method implemented in SIESTA package are used to calculate the electronic states, exchange coupling parameters of an infinite chain model system of Cr_8. We demonstrate how, under opportune modifications to the ring cycle structure, different one-dimensional chain models can be devised, with the capability of mimicking with good approximation the electronic and magnetic properties of the original Cr_8 molecule. Such models offer an unique opportunity, in virtue of the reduced computational effort, to carry out extensive investigations of a whole set of molecules belonging to the Cr-based molecular rings family.

Keywords: Density functional theory · Cr antiferromagnetic rings

1 Introduction

We present in the following a density-functional study of one-dimensional chain model systems, which aim to mimic the electronic and magnetic properties of the Cr_8 antiferromagnetic (AFM) molecular ring. This molecule is the first member of a vast family of cyclic Cr-based organometallic magnets [1] to be synthesized with high yield [2] and attracted much interest in virtue of possible applications in quantum information processing [3,4] and interesting fundamental properties [5–8]. The Cr_8 molecule is characterized by a perfect antiferromagnetic coupling between the eight $s = 3/2$ Cr spins, which lead, at zero field, to an S = 0 ground state. The study of molecules of this size within an all-electron first principle method, as the linearized augmented plane wave (LAPW) technique [9] implemented in the Wien2k simulation package [10], which is our method of choice to study the electronic and magnetic properties of such systems, represents a severe task even for modern parallel computers [11]. Alternatively, fast gaussians-based quantum chemistry packages, such as NWChem could be employed with success [12,13].

The amount of cpu time at disposal for scientific calculations increases year after year, and the optimization of the codes plays also a crucial role in the improvement of scientific computing. On the other side it is of evident importance the possibility to scale down the size of the systems under interest, by devising toy models or smaller replications which eventually could give reliable information and delve the knowledge of the, larger, real systems under interest. The advantages are twofold: computation becomes more feasible and the reduced number of degrees of freedom might help in capturing the underlining the physics, at least on the qualitative level.

Our scope is to show that under opportune conditions, we can devise several one-dimensional models obtained by straightening the cyclic structure, lining up with a certain periodicity the Cr atoms, without altering in a sensible manner the ground state magnetic properties of the original ring molecule.

Alternatively, calculations can be speed up using different methods as well as with the help of different computational packages. We would like to present fairly good results acquired with the help of SIESTA suit [14,15], that uses pseudopotential method [16,17]. In our Siesta calculations exchange and correlation effects are accounted for by the generalized gradient local approximation (GGA) with exchange-correlation potential proposed by Perdew, Burke and Ernzerhof (PBE) [18].

2 Chain Models and Computational Details

The molecular units used to build the chain models are shown in Fig. 1. As discussed in a previous work [19], the model of Fig. 1(a) is built from a single repetition unit of a Cr_8 molecule, i.e. a segment which represents 1/8 of the whole ring structure. All the Cr atoms line up on a single line, and therefore we call this model a "linear" chain. By changing the repeated unit, for instance by keeping only one forth of the whole ring (two Cr-Cr segments) or a larger number of Cr-Cr segments, we could construct different models as depicted in Fig. 1(b) and (c), which somehow contains larger untouched portions of the original molecule. For the segments in Fig. 1(a)–(c) the periodic boundary conditions are imposed in the direction of segments and the coordination symmetry for Cr ions is preserved. The names "zigzag" and "snake" indicates how the Cr atoms are placed around the linear direction along which we will repeat the unit. The assignment of the nearest-neighbour couplings J_i $(i = 1, 2, 3)$ is determined in Fig. 1(a)–(c), too. In order to give an idea on the computational gain we report in Table 1 several parameters characterizing the size of the model systems.

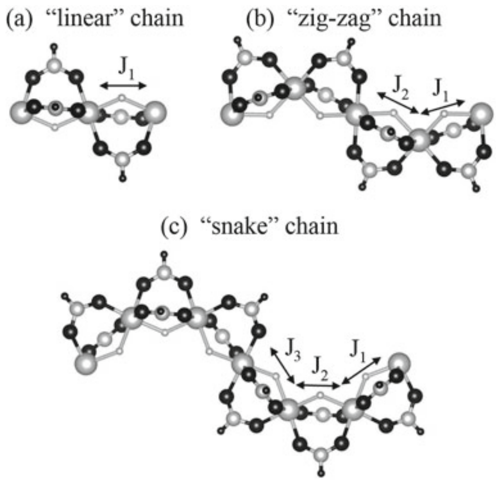


Fig. 1. Simulation units of the three different chain model systems of Cr_8. The size and color representation is following: large gray Cr, medium grey C, small grey F, medium black O and small black H. For details of the structures see the text.

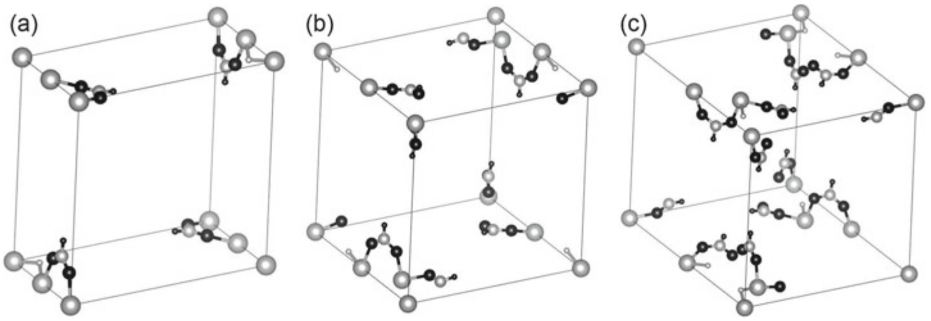


Fig. 2. Computational unit cell of the three different chain model systems of Cr_8. The color representation as in Fig. 1

We have accommodated the repetition units depicted in Fig. 1 into supercells, applying periodic boundary conditions, and considering enough vacuum space to decouple the different images. Computational supercells for “linear”, “zigzag” and “snake” chain models are presented in Fig. 2(a), (b) and (c) respectively. The total number and the number of inequivalent atoms are listed in Table 1 for the chain models as well as for the original Cr_8 molecule. The number of APWs necessary to ensure converged results, as evident from Table 1, is extremely reduced for all the chain models. Well converged magnetic properties can be obtained by using around 100–150 APWs per atom in cell, which corresponds to an energy cutoff of 10–13 Ry depending on the size of the simulation cell. The

Table 1. Comparison between different quantities characterizing the effort to simulate the different type of chain models and the Cr_8 molecule; the N. of atoms in the unit cell, the N. of inequivalent atoms in the unit cell and the N. of cores vs. CPU time needed for a single iteration of the self-consistency cycle (see text for details of the computing machines).

	"linear"	"zigzag"	"snake"	Cr_8
Wien2k Matrix size (N. of APWs)	≈ 3000	≈ 6500	≈ 9000	≈ 15000
Nr of atoms/unit cell	20	40	60	160
Nr of inequivalent atoms/unit cell	11	21	31	20
Wien2k No. of cores/time per 1 iter [min]	1/10	8/10	32/20	64/40
Siesta No. of cores/time per 1 iter [min]	4/0.4	4/2	4/5.4	4/11.1

gain is tremendous for the "linear" chain model, while when we move towards longer and more curved models, as for the "snake" one, the computational gain decreases sensibly. Overall the chain model systems of Cr_8 described above are computationally much more convenient as compared to the original molecule, and under proper modifications, are applicable to several other members of the cyclic Cr-based magnetic molecule family, allowing for a more extensive study of their magnetic properties. In the case of the "snake" model, despite the larger number of inequivalent atoms due to the presence of the inversion symmetry operation alone, one is faced still with a reduction of around 50 % in the number of APWs, which because of the non linear scaling of an eigenvalue problem, means that the "snake" model can be simulated twice as fast with half the number of CPUs than the Cr_8 molecule. The elapsed time, in minute, are relative to simulations performed on a multiprocessors, shared memory, BM BladeCenter LS21 Cluster, composed of 1280 4-way nodes, each containing 2 Opteron dual-core processors, with a clock of 2.4 GHz, 8 GB of memory, and with a global peak performance of 26.6 TFlops. Obviously, increasing further the repetition unit, will lead to a model which does not represent anymore a real advantage in terms of computing effort, compared to the original molecule.

Since pseudpotential-based approach is less computationally demanding than all electron method it allows to obtain results for these models with relatively small computing effort. On the other hand, testing and selection of a suitable pseudopotential is a tedious and time-consuming [20]. The SIESTA computations were performed using just one computer, equipped with 24 GB of memory and 6-core processor with a clock of 2.67 GHz. However only four cores and 1 GB per core were used. SIESTA does not use any symmetry in its calculations whatsoever, apart from the time reversal symmetry for k-points. Therefore the greatest computational gain for chain models is achieved through decreasing the total number of atoms within the super cell. This leads to proportional decrease in the number of atomic orbitals, that are used in computations.

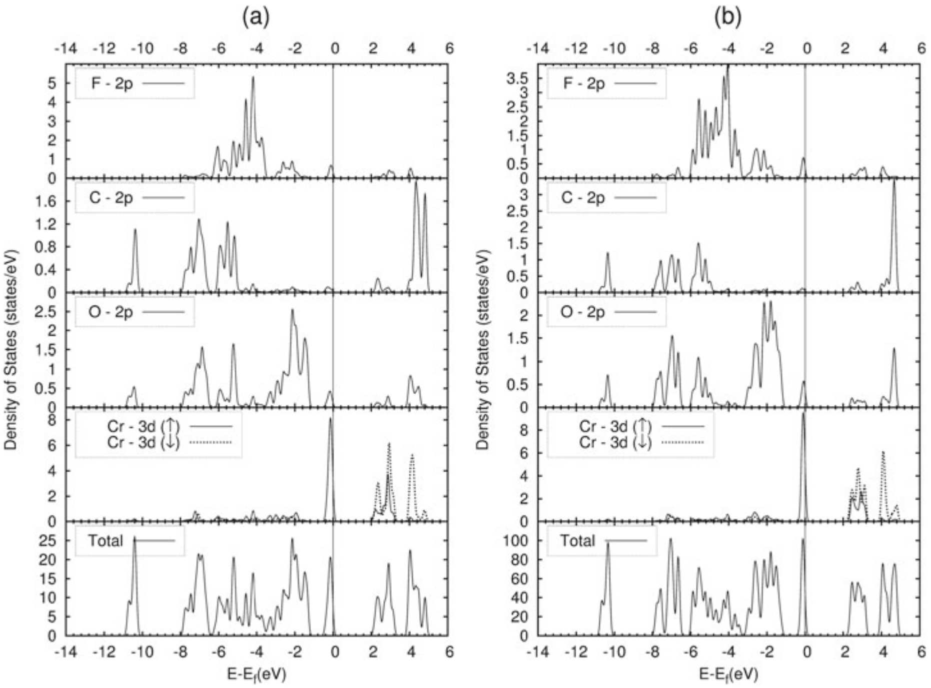


Fig. 3. Total and atom/orbital projected density of states (DOS) of the chain model of Cr_8 (a) and of the Cr_8 molecule (b) in the AFM ground state within the GGA approach and pseudopotential method.

3 Results and Discussion

3.1 "linear" Chain Model

Let us focus for the moment on the 'linear" chain model of Fig. 1(a). Distances and angles between the Cr atoms and the atoms composing the bridges are kept identical within a single Cr-Cr segment, while the order of appearance of the bridges in the second Cr-Cr segment is varied, switching the Cr-F and one of the Cr-O bonds.

In order to investigate whether such bond rearrangement influences the electronic states and the magnetic interaction between the spins located at the Cr atoms we show in Fig. 3(a) and (b) respectively the total density of states (DOS), as well as the partial DOS obtained by the projections onto the Cr(3d), F(2p), C(2p) and O(2p) orbitals, of the "linear" chain model and of Cr_8 original molecule in the AFM ground state configuration (both obtained from Siesta). Similar results for the chain model and Cr_8 molecule calculated from Wien2k can be found in Ref. [19] in Fig. 5.

The density of states of the chain model and of the Cr_8 original molecule are very similar, throughout the whole energy range; a small shift at lower energies is observed for some states, while in the crucial (for the magnetic properties) range around Fermi energy the model works very well. For the results from Siesta package discrepancy at low energies is natural for this method, because only selected valence states are taken into account.

Moreover we find that magnetic moments and spin density distributions in the original molecule and in the model complex are also very similar. In order to test the real effectiveness of the chain model in resembling the magnetic properties of the original ring Cr_8 molecule, the exchange interaction parameters J must be addressed. The unique J parameter is extracted by simply mapping the ab-initio total energies of the antiferromagnetic (ground state) and ferromagnetic states (respectively E^{TOT}_{AFM} and E^{TOT}_{FM}) onto an effective spin Ising Hamiltonian,

$$H = 2J s_1 \cdot s_2 \ , \tag{1}$$

where $s_{1,2} = s = 3/2$ is the spin of the Cr ions, yielding

$$J = \frac{[E^{TOT}_{FM} - E^{TOT}_{AFM}]}{4s^2} \ . \tag{2}$$

For Wien2k within the generalized gradient approximation (GGA) to the exchange-correlation functional (more precisely, the one suggested by Perdew et al. in 1996, i.e. PBE96 [18]), we obtain the value $J_{chain} = 6.2$ meV in qualitative agreement with the value of $J_{Cr_8} = 5.8$ meV obtained for Cr_8 ring [11], while the value inferred by the experiments is $J_{Exp} = 1.5$ meV. Also for SIESTA calculation for chain model we obtain similar value of exchange interaction parameter $J = 6.5$ meV [21]. As discussed in Ref. [11], the overestimation of the exchange parameters testimonies the error of local exchange and correlation functionals in describing the electron-electron correlations in such molecules, as much as it happens in parent chromia bulk oxides [22].

Inclusion of static correlation in the LDA+U method [23] or hybrid functionals [12] improves the agreement with the experiments; the value of J which best fit the experiments is found assuming an Hubbard repulsion term of $U = 5$ eV, which is consistent with the value assumed for Cr_8 in Ref. [11], and with values already used in Cr oxides bulk compounds.

3.2 "zigzag" and "snake" Chain Models

Very similar results to the one discussed above for the simple "linear" chain model are found for the other two models. We supply in Table 2 the exchange interaction parameters that can be extracted by total energy differences between different magnetic configurations, using the corresponding spin Hamiltonian. In the case of the "linear" and original molecule, because of the special symmetry and of the number (two) of inequivalent Cr atoms in the cell, only one J is present, while respectively for the "zigzag" and the "snake" models, two and three exchange parameters can be defined, if one considers only nearest neighbor (*nn*) couplings between the Cr ions (see Fig. 1).

Table 2. Comparison between values of the exchange interaction parameters (in meV) extracted and averaged from the different systems (see Fig. 1 for reference).

	LAPW method				Pseudopotential method			
	"linear"	"zigzag"	"snake"	Cr_8	"linear"	"zigzag"	"snake"	Cr_8
J_1	6.2	6.3	6.2	5.8	6.5	6.0	6.0	6.6
J_2		7.0	7.4			6.9	7.1	
J_3			6.1				6.0	

Moreover the system of equations relating the spin degrees of freedom are over-determined, i.e. there are more equations than variables, and depending on the subset of the magnetic configurations used for extracting the J's, small differences are found. For instance, in the "zigzag" model there are three inequivalent Cr atoms, and we can, upon choosing proper initial conditions, converge towards four different magnetic configurations, i.e. ferromagnetic, antiferromagnetic, and two "ferrimagnetic" solutions, where two subsequent Cr spin moments point into the same direction. Having 4 different spin-distributions, and only three parameters (the two Js and the implicit electronic term) there exist $4!/((4-3)!3!) = 4$ possible subsets, for each of which a set of Js could be extracted. We find in our calculations with LAPW method that J_1 spans values in the range 6.0–6.5 meV, and J_2 spans values in the range 6.8–7.3 meV. Similar reasonings can be carried out for the "snake" model, where having 4 inequivalent atoms, it is possible to converged 8 different magnetic solutions; four parameter (three Js and the implicit electronic term) can be extracted from 4 magnetic solutions only so that $8!/((8-4)!4!) = 70$ possible subsets could be chosen. Again the variations in J_1, J_2 and J_3 are of the same order of magnitudes and amount 0.5 eV. The situation is similar for results obtained with SIESTA. Here we can also observe, that the values of exchange parameters for different chain models are close to each other. It is worth mentioning, that identical tendency was noticed for Cr_8 molecule, when the two different alternating nearest neighbor exchange parameters were considered [14]. The J's reported in Table. 2 for the "zigzag" and "snake" chains are averaged over this subset choices. Though the values of J are still overestimated in comparison to empirical ones, they are in agreement with other theoretical calculations. This suggests that the "snake" and "zig-zag" approximations do not improve the J's estimates.

Both the variation for each Js as well as the different values of the Js, i.e. $J_1 \neq J_2 \neq J_3$ have a twofold origin. First, one is subjected to intrinsic errors when trying to extract the exchange parameters by mapping total energies onto spin Hamiltonians, due to non-complete localization of the spins and to the existence of higher order interactions beyond Ising model between the electrons, which, although of smaller entity, are often present in magnetic systems. Secondly, there is a small error induced by the structural rearrangement concept underlying the physcics of above discussed models. Both these source of error are basically hidden, being of the same order of magnitude as the numerical precision of a whatsoever code based on density-functional methods.

In the over-determined models, we could exploit the exceeding number of possible solutions, in order to check whether next-nearest neighbor (*nnn*) interactions between the Cr ions are present. These type of interactions have not been considered for the Cr_8 molecule, where magnetic and spectroscopic data available could be explained very nicely only with the existence of nearest neighbor coupling. In the "zigzag" model we could add one *nnn* interaction, while in the "snake" model we could had two *nnn* interactions. The "zigzag" model becomes well-determined, and the *nnn* coupling that could be extracted has a value of $J_{nnn} = -0.2\,\text{meV}$ which is more than one order of magnitude smaller than the *nn* Js. Similar values could be extracted, choosing, different subsets of solutions in the "snake" model. In virtue of the discussion above on the error bar associated to the *nn* Js, we could state that, if present, such magnetic interactions between next-nearest neighbor Cr ions are very small in Cr-based cyclic molecular magnets. If this holds for closed Cr homometallic rings (Cr_8, Cr_{10}), we could not exclude the existence of longer range interactions in open ring molecules or in heterometallic wheel complexes, when smaller distance between the paramagnetic ions, different type of bridges as well as eventually larger spin delocalization might play a more important role than what is observed here.

4 Summary and Outlook

To conclude, we have presented a density-functional investigation of possible chain model systems, with different sizes, which are able to describe the electronic and magnetic properties of the original ring-shaped Cr_8 molecule. There is a whole experimental literature describing the family of cyclic molecular magnets, with Cr, and eventually other transition metal ions, as paramagnetic centers. In fact, by varying the experimental conditions, as well as the components of the synthesis reaction, a whole series of homo- and hetero-metallic Cr-based cages have been engineered [For recent reviews on this subject see Refs. [24] and [1]]. Looking into the different molecules belonging to this family, depending on the cage structure, symmetry of the paramagnetic centers and nature of the bridges, it is likely to individuate an atom, with proper coordination, which could act as the inversion center of the curvature, as, for instance, happens in the $Cr_{12}Ni$ seahorse molecule. [24] The lack of symmetry operations, except inversion, can be exploited in the longer chain models for investigating in a systematic manner, for instance, how different bridges, other than carboxylate, can transmit the magnetic interaction between the paramagnetic centers, and also how the attachment of functionalizing molecules or atoms, as sulfur groups, procedure needed in order to graft with success such inorganic molecules to surfaces, [25] influences the type and strength of the magnetic coupling between the paramagnetic ions.

Furthermore we have shown that the linear models give results that are consistent with the method. Also the differences between the results obtained within different packages are insignificant, which means that they are independent on the computational software. Within this context methods based on pseudopotentials seem to be promising for solving problems mentioned above. Especially since they allow for huge computational speed up in conjunction with proposed models.

Acknowledgments. The calculations were performed on computer facilities granted by the CNR-INFM Iniziativa Trasversale Calcolo Parallelo at the CINECA supercomputing center (Italy), Poznan Supercomputing and Networking Center (Poland) as well as within DECI programme by the PRACE-2IP (FP7/2007-2013) under grant agreement no RI-283493. Support from the Polish MNiSW through the grant No N519 579138 is also acknowledged.

References

1. Affronte, M., Carretta, S., Timco, G.A., Winpenny, R.E.P.: A ring cycle: studies of heterometallic wheels. Chem. Commun. **18**, 1789 (2007)
2. van Slageren, J., Sessoli, R., Gatteschi, D., Smith, A.A., Helliwell, M., Winpenny, R.E.P., Cornia, A., Barra, A.L., Jansen, A.G.M., Rentschler, E., Timco, G.A.: Magnetic anisotropy of the antiferromagnetic Ring [$Cr_8F_8Piv_{16}$]. Chem. Eur. J. **8**, 277 (2002)
3. Meier, F., Levy, J., Loss, D.: Quantum computing with spin cluster qubits. Phys. Rev. Lett. **90**, 047901 (2003)
4. Troiani, F., Ghirri, A., Affronte, M., Carretta, S., Santini, P., Amoretti, G., Piligkos, S., Timco, G., Winpenny, R.E.P.: Molecular engineering of antiferromagnetic rings for quantum computation. Phys. Rev. Lett. **94**, 207208 (2005)
5. Baker, M.L., Timco, G.A., Piligkos, S., Mathieson, J.S., Mutka, H., Tuna, F., Kozowski, P., Antkowiak, M., Guidi, T., Gupta, T., Rath, H., Woolfson, R.J., Kamieniarz, G., Pritchard, R.G., Weihe, H., Cronin, L., Rajaraman, G., Collison, D., McInnes, E.J.L., Winpenny, R.E.P.: A classification of spin frustration in molecular magnets from a physical study of large oddnumbered-metal, odd electron rings. Proc. Natl. Acad. Sci. **109**, 19113 (2012)
6. Baker, M.L., Guidi, T., Carretta, S., Ollivier, J., Mutka, H., Gudel, H.U., Timco, G.A., McInnes, E.J.L., Amoretti, G., Winpenny, R.E.P., Santini, P.: Spin dynamics of molecular nanomagnets unravelled at atomic scale by four-dimensional inelastic neutron scattering. Nat. Phys. **8**, 906 (2012)
7. Kozłowski, P., Kamieniarz, G., Antkowiak, M., Tuna, F., Timco, G.A., Winpenny, R.E.P.: Phenomenological modeling of the anisotropic molecular-based ring Cr7Cd. Polyhedron **28**, 1852 (2009)
8. Antkowiak, M., Kozowski, P., Kamieniarz, G., Timco, G.A., Tuna, F., Winpenny, R.E.P.: Detection of ground states in frustrated molecular rings by the in-field local magnetization profiles. Phys. Rev. B **87**, 184430 (2013)
9. Andersen, O.K.: Linear methods in band theory. Phys. Rev. B **12**, 3060 (1975)
10. Blaha, P., Schwarz, K., Madsen, G., Kvasnicka, D., Luitz, J.: WIEN2k, An Augmented Plane Wave + Local Orbitals Program for Calculating Crystal Properties. Techn. Universität Wien, Austria (2001). (K. Schwarz, Techn. Universität Wien, Austria)
11. Bellini, V., Olivieri, A., Manghi, F.: Density-functional study of the Cr_8 antiferromagnetic ring. Phys. Rev. B **73**, 184431 (2006)

12. Bellini, V., Affronte, M.: A density-functional study of heterometallic Cr-based molecular rings. J. Phys. Chem. B **114**, 14797 (2010)
13. Bellini, V., Lorusso, G., Candini, A., Wernsdorfer, W., Faust, T.B., Timco, G.A., Winpenny, R.E.P., Affronte, M.: Propagation of spin information at supramolecular scale through hetero-aromatic linkers. Phys. Rev. Lett. **106**, 227205 (2011)
14. Ślusarski, T., Brzostowski, B., Tomecka, D., Kamieniarz, G.: Electronic structure and magnetic properties of a molecular octanuclear chromium-based ring. J. Nanosci. Nanotechnol. **11**, 9080 (2011)
15. Brzostowski, B., Lemański, R., Ślusarski, T., Tomecka, D., Kamieniarz, G.: Chromium-based rings within the DFT and FalicovKimball model approach. J. Nanopart. Res. **15**, 1528 (2013)
16. Ordejón, P., Artacho, E., Soler, J.M.: Self-consistent order-N density-functional calculations for very large systems. Phys. Rev. B (Rapid Comm.) **53**, R10441 (1996)
17. Soler, J.M., Artacho, E., Gale, J.D., García, A., Junquera, J., Ordejón, P., Sánchez-Portal, D.: The siesta method for ab initio order-N materials simulation. J. Phys. Condens. Matter **14**, 2745 (2002)
18. Perdew, J.P., Burke, K., Ernzerhof, M.: Generalized gradient approximation made simple. Phys. Rev. Lett. **77**, 3865 (1996)
19. Tomecka, D., Bellini, V., Manghi, F., Affronte, M., Kamieniarz, G.: Ab-initio study on a chain model of the Cr_8 molecular magnet. Phys. Rev. B **77**, 224401 (2008)
20. Brzostowski, B., Ślusarski, T., Kamieniarz, G.: DFT study of octanuclear molecular chromium-based ring using new pseudopotential parameters. Acta Phys. Pol. A **121**, 1115 (2012)
21. Ślusarski, T., Brzostowski, B., Tomecka, D., Kamieniarz, G.: Application of the package SIESTA to linear models of a molecular chromium-based ring. Acta Phys. Pol. A **118**, 967 (2010)
22. Rohrbach, A., Hafner, J., Kresse, G.: Ab initio study of the (0001) surfaces of hematite and chromia: influence of strong electronic correlations. Phys. Rev. B **70**, 125426 (2004)
23. Liechtenstein, A.I., Katnelson, M.I., Antropov, V.P., Gubanov, V.A.: Local spin density functional approach to the theory of exchange interactions in ferromagnetic metals and alloys. J. Magn. Magn. Materials **67**, 65 (1987)
24. McInnes, E.J.L., Piligkos, S., Timco, G.A., Winpenny, R.E.P.: Studies of chromium cages and wheels. Coord. Chem. Rev. **249**, 2577 (2005)
25. Corradini, V., Biagi, R., del Pennino, U., Renzi, V.D., Gambardella, A., Affronte, M., Muryn, C.A., Timco, G.A., Winpenny, R.E.P.: Isolated heterometallic Cr_7Ni rings grafted on Au(111) surface. Inorg. Chem. **46**, 4937 (2007)

Non-perturbative Methods in Phenomenological Simulations of Ring-Shape Molecular Nanomagnets

Piotr Kozłowski[1], Grzegorz Musiał[1(✉)], Monika Haglauer[1], Wojciech Florek[1], Michał Antkowiak[1], Filippo Esposito[2], and Dante Gatteschi[3]

[1] Faculty of Physics, Adam Mickiewicz University, ul. Umultowska 85, PL-61-614 Poznań, Poland
gmusial@amu.edu.pl
[2] Dipartimento di Scienze Fisiche, Universitá di Napoli, Piazzale V. Tecchio 80, I-80125 Napoli, Italy
[3] Department of Chemistry, INSTM and Universitá di Firenze, via della Lastruccia 3, I-50019 Sesto Fiorentino, Italy

Abstract. Two non-perturbative numerically exact methods: exact diagonalization and quantum transfer matrix are applied to computationally complex Heisenberg-like spin models of ring shaped molecular nanomagnets and implemented in the high performance computing environment. These methods are applicable to the wide class of ring-shaped nanomagnets. For the hypothetical antiferromagnetic nanomagnet Ni_{12} the influence of single-ion anisotropy on the ground states is investigated. For Cr_8 it is demonstrated that the alternation of the nearest-neighbor bilinear exchange couplings leads to small changes in the magnetic torque with respect to the uniformly coupled system. Specific heat and entropy for Cr_8 are showed to be good indicators of crossing fields. The applicability of the Lande rule to both systems is checked.

Keywords: Molecular nanomagnet · Quantum transfer matrix · Exact diagonalization · Heisenberg Hamiltonian · Magnetic torque · High performance computing

1 Introduction

Molecular clusters have been in focus for more than thirty years [1] attracting a lot of scientific interest and giving rise to the emerging field of molecular nanotechnology. Many scientific research in this field was motivated by foreseen applications e.g. in storage media and hybrid or quantum computers [2]. Precise modeling of such systems plays thus a very important role and is a part of common effort which constantly works out practical applications of molecular nanomagnets [3].

A very interesting class of molecular clusters are nanomagnets which contain a core made of transition-metal ions. These magnetic molecules are shielded from one another by a shell of organic ligands and therefore behave like separate quantum magnets. Since their quantum microscopic properties can be observed on a macroscopic scale they are of interest both, from theoretical and experimental point of view.

Phenomenological modelling of such molecules is based on the assumption that the intercluster interactions are negligible and that the properties of a system are determined by strong intracluster exchange coupling. In the first approximation a pure Heisenberg Hamiltonian is used [4]. However, the experimental findings indicate more complex interactions, suggesting inclusion of non-Heisenberg terms to the Hamiltonian. A standard approach to these more complex models is based on perturbative techniques [5], which however are approximate and have a limited area of application.

In this paper two non-perturbative, numerically exact methods: exact diagonalization (ED) [6] and quantum transfer matrix (QTM) [7] are implemented in the high performance computing environment and applied to simulate ring shaped molecular nanomagnets Ni_{12} and Cr_8. We consider more complex interactions in the Hamiltonian than those used in the literature discussed above.

2 Spin Models and Thermodynamic Quantities

We focus on the applications of our methods to the ring-shape molecular nanomagnets $Ni_{12}(Ni_{12}(O_2CMe)_{12}(chp)_{12}(H_2O)_6(thf)_6)$ [8], Cr_8 $(Cr_8F_8Piv_{16})$ [4], Cr_9 [9,10], V_8 and V_{10} [11] (chp = 6-chloro-2-pyridonate, thf = tetrahydrofuran, Piv = pivalate). The first two nanomagnets are modelled in this work. Ni_{12} has the properties of a single molecular magnet (SMM), i.e. large spin in the ground state and zero field splitting (ZFS) of the low laying states. It can be modelled with $s = 1$ quantum spin ring with the prevailing ferromagnetic exchange interactions [12]. In this paper we assume antiferromagnetic couplings and study the dependence of the energy spectrum on the single-ion anisotropy by means of ED. In particular we analyze the pattern of level crossings. Since there is no Ni_{12} compound with antiferromagnetic nearest neighbor interactions we do not compare our results with the experiment and our study has up to now a purely theoretical significance. It can however reveal the relevance of the anisotropic effects for magnetic and thermodynamic properties of molecular magnets.

The chromium molecule can be modeled with the 8 membered $s = 3/2$ antiferromagnetically coupled quantum spin ring. In the previous investigations [4,13] the nearest neighbor exchange coupling and single ion anisotropy were determined on the basis of inelastic neutron scattering (INS), electron paramagnetic resonance and susceptibility measurements. They were also confirmed by fitting to specific heat data [13]. The symmetry of the molecule suggests the possibility of the alternation of the couplings [14], but neither fitting to the INS data [13] nor the calculations for the susceptibility and specific heat [15] are able to detect possible bond alternation. Here we analyze sensitivity of the low-temperature magnetic torque to the bond alternation of the couplings. We calculate also the low temperature specific heat and entropy as functions of the magnetic field. The calculations are performed by means of the QTM technique.

To model the molecular rings we use the following quantum spin ($s = 3/2$ for Cr_8 and $s = 1$ for Ni_{12}) Hamiltonian:

$$\mathcal{H} = \sum_{j=1}^{n/2} \left(J^{\mathrm{o}} \boldsymbol{s}_{2j-1} \cdot \boldsymbol{s}_{2j} + J^{\mathrm{e}} \boldsymbol{s}_{2j} \cdot \boldsymbol{s}_{2j+1} \right) + \sum_{j=1}^{n} \left(D(s_j^z)^2 - g\mu_{\mathrm{B}} B \left(s_j^x \sin\theta + s_j^z \cos\theta \right) \right) , \tag{1}$$

where $J^{\mathrm{o,e}}$ are nearest-neighbor exchange integrals for 'odd' and 'even' pairs, respectively, n is the number of sites ($n = 8$ for Cr_8 and $n = 12$ for Ni_{12}), D is the (site-independent) single-ion anisotropy, B is the external magnetic field applied in the $x - z$ plane and forming an angle θ with the z axis, g is the corresponding Landé factor and μ_{B} stands for Bohr magneton. Periodic boundary conditions ($j + n \equiv j$) are assumed due to the ring geometry.

Starting from the free energy

$$F = -k_{\mathrm{B}} T \ln Z \ , \qquad Z = \mathrm{Tr} e^{-\beta \mathcal{H}}, \tag{2}$$

one can calculate the thermodynamic functions of interest, like e.g. the specific heat C, the magnetic susceptibility χ and the entropy S by numerical differentiation of F, or by using the first two moments of the total spin operators

$$\chi = \lim_{m \to \infty} \beta (g\mu_{\mathrm{B}})^2 \left(\langle (S^z)^2 \rangle - \langle S^z \rangle^2 \right) \tag{3}$$

$$\tau_y = -g\mu_{\mathrm{B}} B \left(\langle S^x \rangle \cos\theta - \langle S^z \rangle \sin\theta \right) \ . \tag{4}$$

$\langle ... \rangle$ stands for thermal average and $\boldsymbol{\tau}$ is a magnetic torque defined as a vector product of the averaged total magnetization $\boldsymbol{M} = \sum_{i=1}^{n} \langle \boldsymbol{s}_i \rangle$ and the magnetic field $\boldsymbol{B}$

$$\boldsymbol{\tau} = \boldsymbol{M} \times \boldsymbol{B} \ . \tag{5}$$

Since the magnetic field is applied in the $x - z$ plane the torque has only y-th component defined by equation (4). We notice that the last quantity is very sensitive to the anisotropy. From the definition (Eq. 5) it is equal to zero if anisotropy is vanishing because then external magnetic field is collinear with the total magnetization.

3 Numerical Methods

In exact diagonalization technique only non-alternating couplings are taken into account. The diagonalization is done numerically and exploits the symmetries of the Hamiltonian, which is invariant with respect to the renumbering of sites $(1,\ 2,\ \ldots\ ,\ N-1,\ N) \to (2,\ 3,\ \ldots\ ,\ N,\ 1)$ generated by the shift operator

$$\mathcal{P} \equiv \sum_{s_1^z} \ldots \sum_{s_N^z} \mid s_2^z \ldots s_N^z s_1^z \rangle \langle s_1^z s_2^z \ldots s_N^z \mid , \tag{6}$$

and the mirror reflection corresponding to the transformation $(1,\ 2,\ \dots\ ,\ N-1,\ N) \to (N,\ N-1,\ \dots\ ,\ 2,\ 1)$. As a result the eigenstates of the Hamiltonian are classified by three quantum numbers (S^z, k, r), where $-ns \le S^z \le ns$ is the z component of total spin, $0 \le k \le n/2$ corresponds to translational invariance and $r = \pm 1$ is related to the mirror reflection.

The quantum transfer matrix technique applies the Trotter formula to the exponent of the Hamiltonian (1) expressed as a sum of two non-commuting Hamiltonians $\mathcal{H}^{\mathrm{o}}$ and $\mathcal{H}^{\mathrm{e}}$. Then the partition function can be written as

$$Z = \lim_{m\to\infty} Z_m = \lim_{m\to\infty} \mathrm{Tr}\left(\mathrm{e}^{-\frac{\beta}{m}\mathcal{H}^{\mathrm{o}}}\mathrm{e}^{-\frac{\beta}{m}\mathcal{H}^{\mathrm{e}}}\right)^m . \tag{7}$$

The symmetries of the $\mathcal{H}^{\mathrm{o}}$ and $\mathcal{H}^{\mathrm{e}}$ allow us to express Z_m with the help of a single sparse matrix V and the shift operators $\mathcal{P}$, $\mathcal{P}^\dagger$ ([7] and references therein)

$$Z_m = \mathrm{Tr}\left[\left(V^{\mathrm{o}}\mathcal{P}^\dagger\mathcal{P}^\dagger\right)^{n/2}\mathcal{P}^\dagger\left(V^{\mathrm{e}}\mathcal{P}^\dagger\mathcal{P}^\dagger\right)^{n/2}\mathcal{P}\right]^m . \tag{8}$$

The trace in (8) goes over 4^8 dimensional spin space. In a similar way one can calculate any m-th approximant $\langle A\rangle_m$ of a quantity A such as e.g. total spin moments $\langle S^x\rangle$, $\langle S^z\rangle$ and $\langle (S^z)^2\rangle$ which are used in equations (3) and (4).

$$\langle A\rangle_m = \mathrm{Tr}A\left[\left(V^{\mathrm{o}}\mathcal{P}^\dagger\mathcal{P}^\dagger\right)^{n/2}\mathcal{P}^\dagger\left(V^{\mathrm{e}}\mathcal{P}^\dagger\mathcal{P}^\dagger\right)^{n/2}\mathcal{P}\right]^m / Z_m . \tag{9}$$

The thermodynamic quantities are then calculated for different values of the Trotter index m and extrapolated to $m \to \infty$. It is worth noting that the trace in equations (8) and (9) is calculated by subsequent matrix-vector multiplications. Due to the sparsity of the matrices involved such operation is numerically very efficient and does not require much computer memory.

Both methods have been parallelized and their high efficiency of processing have been demonstrated in real-life computing on several high performance multicomputers [16].

4 Results

We have applied the exact diagonalization technique to the evenly and antiferromagnetically coupled $s = 1$ spin ring of 12 sites which models the hypothetical Ni_{12} nanomolecule and we have calculated the energy levels as a function of the uniform magnetic field B applied along the z axis. The single-ion anisotropy was included and fixed to the value $D/|J| = 0.1$. The presence of positive anisotropy substantially modifies the energy levels structure of this molecule [12]. Like in the case of negative anisotropy one can observe that with increasing magnetic field B magnetization increases in a characteristic step-like manner. In Fig. 1 the lowest energy levels are presented as a function of the magnetic field. The magnetization steps begin precisely at the so called crossing fields at which the lowest energy levels intersect. For each energy level in Fig. 1 magnetization $M = -S$ with S shown in the legend box. Although, S is not a good quantum number

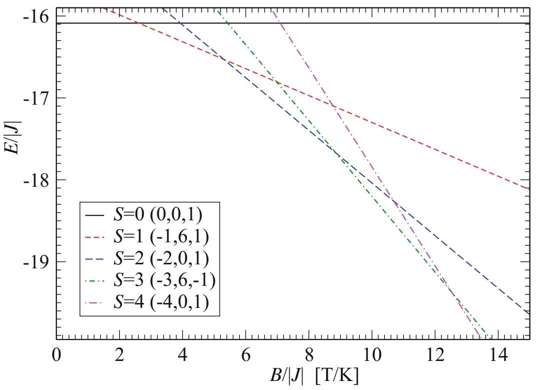


Fig. 1. The magnetic field $B/|J|$ dependence of the low lying energy spectrum for the uniformly antiferromagnetically coupled $s = 1$ spin ring of 12 sites with the single-ion anisotropy $D/|J| = 0.1$

at non-zero D values (see the next paragraphs), but for small $D/|J|$ values it is still numerically applicable, as S values remain close to ones indicated in the legend box. We can group E levels as multiplets of the approximate total spin. For our model we have that the level with $S = 1$ becomes the ground state at $B/|J| = 2.61$; the level with $S = 2$ at $B/|J| = 5.30$; the level with $S = 3$ at $B/|J| = 8.84$ and the level with $S = 4$ at $B/|J| = 12.54$. In comparison to the model with negative anisotropy [15,17] the crossing fields are much bigger leading to much fewer magnetization steps in the same range of the magnetic field. Nevertheless at least the first few crossing fields are still experimentally accessible.

In many finite quantum spin systems the so-called Landé band [18,19] can be observed. Let $E_0(S)$ denote the minimum energy for all states with a given total spin number S, which is a good quantum number in the absence of the anisotropy. The classical Landé rule for a system with even number n of spins s says that gaps $\Delta_S = E_0(S) - E_0(S-1)$, $S = 1, 2, \ldots, ns$, form an arithmetic series: $\Delta_S = S\Delta_1$. Therefore $E_0(S) - E_0(0) = \Delta_1 S(S+1)/2$, where, according to the Lieb-Mattis theorem [20], $E_0(0)$ is the ground state energy. It can be shown that for n even $\Delta_{ns} = 4s$, so Δ_1 should be equal to $4/n$ and the ground state energy $E_0(0) = -s(ns+2)$. However, it is satisfied for $n = 4$ only and in the classical limit $s \to \infty$ [21], so hereafter this value is denoted as Δ_1^{c} and referred to as the classical gap. For small n and large s the Landé rule is satisfied approximately, in particular $\Delta_S \approx S\Delta_1$ for small S, even for $\Delta_1 \neq \Delta_1^{\mathrm{c}}$. This behavior, sometimes called 'the quantum Landé rule', is often used to determine thermodynamic properties at low temperatures [13].

Taking into account also the results received for six spins $s = 3/2$ (unpublished) and $s = 5/2$ [22] we have calculated three parameters: the ratio $\Delta_1^{\mathrm{c}}/\Delta_1$, $E_0(ns) - E_0(0)$ compared to the classical value $2s(ns+1)$, and the ratio $\Delta_S/S\Delta_1$ for a couple values of S; the results are collected in Table 1.

Table 1. Comparision of exact diagonalization results with the classical and quantum Landé rules (all figures in %; see text for details)

n	s	$\frac{\varDelta_1^c}{\varDelta_1}$	$\frac{2s(ns+1)}{E_0(ns)-E_0(0)}$	$\frac{\varDelta_2}{2\varDelta_1}$	$\frac{\varDelta_3}{3\varDelta_1}$	$\frac{\varDelta_4}{4\varDelta_1}$	$\frac{\varDelta_5}{5\varDelta_1}$
6	5/2	96.382	98.238	99.952	99.876	99.759	99.602
6	3/2	94.496	97.110	99.965	99.773	99.380	98.685
8	3/2	89.510	95.285	99.803	99.051	98.214	97.186
12	1	68.842	90.060	88.393	86.142	83.937	81.764

The classical Landé rule is quite well satisfied in the 'classical' limits, i.e. for large s ($s \to \infty$) and for small n ($n \to 4$). When $n = 12$ and $s = 1$ the calculated gap $\varDelta_1 \approx 0.484 \approx 1.5\varDelta_1^c$. However, the gap $E_0(ns) - E_0(0)$ does not differ from the classical value so dramatically. It is caused by the fact that for larger S $\varDelta_S/S\varDelta_1^c$ tends to 1 which is reached at $S = ns$. It confirms that the quantum effects are most important near the ground state of the isotropic Heisenberg antiferromagnet (small S). The values of ratios $\varDelta_S/S\varDelta_1$ for $S = 2, 3, 4, 5$ in three top lines show that the qunatum Landé rule can be used for systems with small n and large s. On the other hand, the fourth line indicates that for large systems with small spin number s the Landé rule gives a very rough approximation of the thermodynamic properties and, for example, the estimated values of the exchange integrals have very low precision (at most two decimal digits).

Using the quantum transfer matrix technique we have calculated the low temperature magnetic torque (Fig. 2) the magnetic specific heat (Fig. 3) and the entropy (Fig. 4) for the Cr_8 ring with the parameters determined by INS in [13] ($J^o = J^e = 16.94\,K$ and $D = -0.34\,K$) as the parameters from DFT calculations are much less accurate [23, 24]. Our results obtained for the torque qualitatively agree with the theoretical predictions found in [14] for the same parameters, but differ as concerns the precise values. The almost perfect match is obtained when we rescale our data by a factor 1.46. This difference may be due to the negligence on our side of exchange anisotropy, which in [14] is taken into account by means of the point-dipole approximation. Both theoretical results do not reproduce well details of the experimental curve, namely the precise position of the crossing fields and the character of the steps, which are less steep in the experiment (cf. Fig. 2 in [14]).

The effect of bond alternation ($\varDelta J = |J^o - J^e| > 0$) is rather small (see Fig. 2). We keep the mean value of the couplings $\bar{J} = (J^o + J^e)/2 = 16.94\,K$ and $\varDelta J \leq 0.4\bar{J}$ since only then a good fit to the susceptibility data is obtained [15]. It is worth noting that when the couplings are alternated the crossing fields (i.e. the magnetic fields at which the torque drops) shift towards higher values (see the inset in Fig. 2), which fits better the experimental results obtained in [14] (see Fig. 2 in [14]). Yet, a better fitting would probably require introduction of exchange anisotropy.

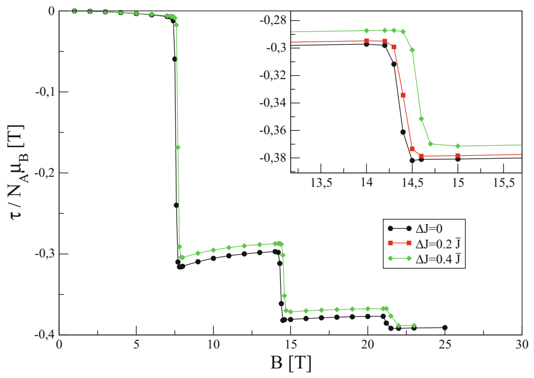


Fig. 2. The magnetic torque τ of Cr_8 for $T = 50\,mK$, $\bar{J} = 16.94\,K$ and $\theta = 6$. The lines serve as a guide for the eye

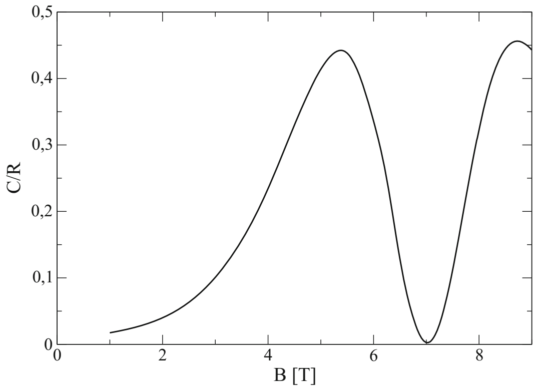


Fig. 3. The magnetic specific heat C of Cr_8 for $T = 0.9\,K$, $J^{\mathrm{o}} = J^{\mathrm{e}} = 16.94\,K$ and $\theta = 65°$

The low temperature specific heat as a function of the magnetic field is shown in Fig. 3. In this case only a qualitative comparison is possible, since the results in the literature [14], both experimental and theoretical, are given in arbitrary units. We can only say that the position of the minimum at $B = 7\,T$, related to the level crossing, agrees precisely with our result. Also the shape of the C curve resembles that of the experiment. We emphasize that due to the high precision of our method, we could determine that the value of C for $B = 7\,T$ is very small ($C/R = 0.0027 \pm 0.0001$) but is not vanishing. The influence of bond alternation on specific heat was investigated in our previous paper [15] and was found to be negligible as long as $\bar{J}$ is kept constant and $\varDelta J \leq 0.4\bar{J}$.

We have also calculated the entropy for the same parameters as the specific heat (Fig. 4). With increasing field an unusual rise of the entropy can be observed with a maximum at $B = 7\,T$. As this maximum precisely corresponds to the

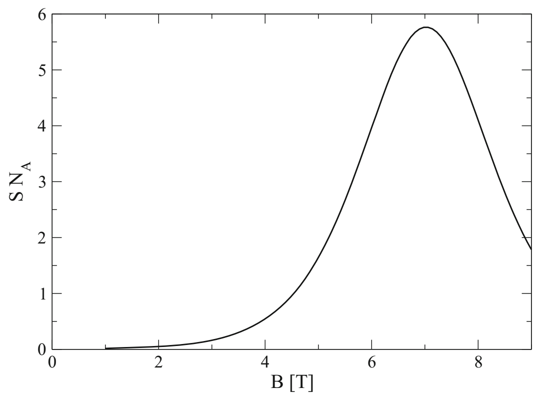


Fig. 4. The entropy S of Cr_8 for $T = 0.9\,K$, $J^{\mathrm{o}} = J^{\mathrm{e}} = 16.94\,K$ and $\theta = 65^{\circ}$

minimum of the specific heat (cf. Fig. 3) it can be related to the energy level crossing [14]. When the separation between two low-lying energy levels decreases the number of accessible microscopic states increases leading to larger values of the entropy. Thus, the analysis of this quantity may be considered as another way to determine the values of crossing or anti-crossing fields.

To obtain the results in a real time we used a parallelized version of our Fortran code. The most demanding were calculations of the torque at $T = 50\,mK$. One point in Fig. 2 required about 20 days of sequential CPU time. Thanks to the application of the high performance parallel computing we reduced this time to about 15 h (with 32 CPU cores engaged).

5 Conclusions

The ED and QTM techniques exploited here give precise, reliable results also for complex systems for which the perturbative methods are usually used. Both techniques have been parallelized with high efficiency, implemented in high performance computing environment [16] and can be applied to a number of existing ring-shape molecular nanomagnets Ni_{12} [8], Cr_8 [4], Cr_9 [9], V_8 and V_{10} [11] and their analogues. The first two ones are modelled in this work.

Using the ED technique we discovered that the change of the anisotropy sign modifies significantly the energy structure of the anisotropic, antiferromagnetically coupled Ni_{12} spin ring placed in the magnetic field. As a result the crossing fields have different distribution leading to more sparse magnetization steps induced by the field. This finding indicates importance of the single-ion anisotropy in antiferromagnetic metal rings for the energy level structure, which influences their thermodynamic properties.

We have shown that large antiferromagnetic systems with small spins ($s \leq 5/2$) do not satisfy the Landé rule, even in its 'quantum' version, especially near the ground state (for small total spin S). This problem had been previously discussed by Engelhardt and Luban [21] and we confirmed that for $n > 10$ deviation from the Landé rule cannot be neglected. It has to be stressed that the Quantum Monte Carlo method gives large numerical errors when $S < ns/10$ [21], so the first energy gap Δ_1 can be precisely determined only for small systems. With the exact diagonalization methods presented here, we can calculate Δ_1 also for significantly larger systems.

Exploiting the QTM method we demonstrated that the bond alternation in Cr_8 model leads to small changes in the magnitude of torque, however the positions of the steps are shifted towards the higher values. This fits better the experimental findings. Nevertheless the fitting is not perfect since the experimental torque steps are wider and lower than those obtained from our calculations. The reasons may be related both, to the experimental setup and to the negligence of some microscopic parameters, like e.g. exchange anisotropy.

By exact calculation of the low temperature specific heat (for Cr_8) as a function of the magnetic field we confirmed the location of the first minimum and found that the precise value of C at this point is very small but not vanishing as suggested earlier. Our results for the entropy imply the possibility of using this quantity to determine positions of the crossing or anti-crossing fields.

Acknowledgments. This work was supported in part by the MNiSW within the project No. N519 579138. Numerical calculations were carried out on the platforms of the Supercomputing and Networking Center in Poznań and of the Academic Computer Center in Gdańsk. Part of simulations was also performed on multicomputer pearl in Faculty of Physics at Adam Mickiewicz University.

References

1. McInnes, E.J.L., Piligkos, S., Timco, G.A., Winpenny, R.E.P.: Studies of chromium cages and wheels. Coord. Chem. Rev. **249**, 2577 (2005)
2. Meier, F., Levy, J., Loss, D.: Quantum computing with antiferromagnetic spin clusters. Phys. Rev. B **68**, 134417 (2003)
3. Blagg, R.J., et al.: Magnetic relaxation pathways in lanthanide single-molecule magnets. Nature Chem. **5**, 673 (2013)
4. Baker, M.L., et al.: Spin dynamics of molecular nanomagnets unravelled at atomic scale by four-dimensional inelastic neutron scattering. Nat. Phys. **8**, 906 (2012)
5. Liviotti, E., Carretta, S., Amoretti, G.: S-mixing contributions to the higher-order anisotropy terms in the effective spin Hamiltonian for magnetic clusters. J. Chem. Phys. **117**, 3361 (2002)
6. Kamieniarz, G., Matysiak, R., Florek, W., Wałcerz, S.: Characterization of some mesoscopic rings: simulation techniques. J. Magn. Magn. Mat. **203**, 271 (1999)
7. Kamieniarz, G., Matysiak, R.: Transfer matrix simulation technique: effectiveness and applicability to the low-dimensional magnetic systems. J. Comput. Appl. Math. **189**, 471 (2006)
8. Andres, H., et al.: Studies of a nickel-based single-molecule magnet. Chem. Eur. J. **8**, 4867 (2002)
9. Baker, M.L., et al.: A classification of spin frustration in molecular magnets from a physical study of large odd-numbered-metal, odd electron rings. Proc. Natl. Acad. Sci. U.S.A. **109**, 19113 (2012)

10. Antkowiak, M., Kozlowski, P., Kamieniarz, G., Timco, G.A., Tuna, F., Winpenny, R.E.P.: Detection of ground states in frustrated molecular rings by the in-field local magnetization profiles. Phys. Rev. B **87**, 184430 (2013)
11. Laye, R.H., Murrie, M., Ochsenbein, S., Bell, A.R., Teat, S.J., Raftery, J., Güdel, H.-U., McInnes, E.J.L.: Solvothermal syntheses of high-nuclearity vanadium(iii) clusters. Chem. Eur. J. **9**, 6215 (2003)
12. Kamieniarz, G., Haglauer, M., Musiał, G., D'Auria, A.C., Esposito, F., Gatteschi, D.: Single-ion anisotropy effects on the energy spectra of spin $S = 1$ Heisenberg ring. Inorg. Chim. Acta **360**, 3941 (2007)
13. Carretta, S., et al.: Microscopic spin Hamiltonian of a Cr_8 antiferromagnetic ring from inelastic neutron scattering. Phys. Rev. B **67**, 094405 (2003)
14. Carreta, S., Santini, P., Amoretti, G., Affronte, M., Ghirri, A., Sheikin, I., Piligkos, S., Timco, G., Winpenny, R.E.P.: Topology and spin dynamics magnetics in molecules. Phys. Rev. B **72**, 060403(R) (2005)
15. Kamieniarz, G., Kozłowski, P., Musiał, G., Florek, W., Antkowiak, M., Haglauer, M., D'Auria, A.C., Esposito, F.: Phenomenological modeling of molecular-based rings beyond the strong exchange limit: bond alternation and single-ion anisotropy effects. Inorg. Chim. Acta **361**, 3690 (2008)
16. Kozłowski, P., Musiał, G., Antkowiak, M., Gatteschi, D.: Effective parallelization of quantum simulations: nanomagnetic molecular rings. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2013, Part II. LNCS, vol. 8385, pp. 418–427. Springer, Heidelberg (2014)
17. Antkowiak, M., Kozłowski, P., Musiał, G., Florek, W., Kamieniarz, G., Esposito, F.: Modeling of the experimental molecular-based ring-shaped nanomagnets. Acta Phys. Polon. A **118**, 965 (2010)
18. Schnack, J., Luban, M.: Rotational modes in molecular magnets with antiferromagnetic Heisenberg exchange. Phys. Rev. B **63**, 014418 (2000)
19. Waldmann, O.: Spin dynamics of finite antiferromagnetic Heisenberg spin rings. Phys. Rev. B **65**, 024424 (2001)
20. Lieb, E., Mattis, D.: Ordering energy levels of interacting spin systems. J. Math. Phys. **3**, 749 (1962)
21. Engelhardt, L., Luban, M.: Low-temperature magnetization and the excitation spectrum of antiferromagnetic Heisenberg spin rings. Phys. Rev. B **73**, 054430 (2006)
22. Florek, W., Bucikiewicz, S.: Néel probability and spin correlations in some nonmagnetic and nondegenerate states of the hexanuclear antiferromagnetic ring Fe_6: Application of algebraic combinatorics to finite Heisenberg spin systems. Phys. Rev. B **66**, 024411 (2002)
23. Tomecka, D.M., Bellini, V., Troiani, F., Manghi, F., Kamieniarz, G., Affronte, M.: Ab initio study on a chain model of the Cr8 molecular magnet. Phys. Rev. B **77**, 224401 (2008)
24. Brzostowski, B., Lemański, R., Ślusarski, T., Tomecka, D., Kamieniarz, G.: Chromium-based rings within the DFT and Falicov-Kimball model approach. J. Nanopart. Res. **15**, 1528 (2013)

Non-uniform Quantum Spin Chains: Simulations of Static and Dynamic Properties

Artur Barasiński[1], Bartosz Brzostowski[1](✉), Ryszard Matysiak[2], Paweł Sobczak[3], and Dariusz Woźniak[1]

[1] Institute of Physics, University of Zielona Góra, Prof. Z. Szafrana 4A, 65-516 Zielona Góra, Poland
B.Brzostowski@if.uz.zgora.pl

[2] Institute of Engineering and Computer Education, University of Zielona Góra, Prof. Z. Szafrana 4, 65-516 Zielona Góra, Poland

[3] Faculty of Physics, A. Mickiewicz University, ul. Umultowska 85, 61-614 Poznań, Poland

Abstract. Since magnetic materials are often composed of magnetically isolated chains, their magnetic properties can be described by the one-dimensional quantum Heisenberg model. The quantum transfer matrix (QTM) method based on a checkerboard structure has been applied for quantum alternating spin chains. To increase the length of the transfer matrix in the Trotter direction we apply the density-matrix renormalization technique and check the efficiency of parallelization for a part of the code: the construction of the transfer matrix. Next, using the Matrix Product State representation, the time evolution of the ground-state magnetization has been performed after the sudden change in applied field.

Keywords: Static and dynamic magnetic properties · Heisenberg model · Numerical simulations · Parallel processing

1 Introduction

The main features of magnetic properties of many compounds can be adequately described within the framework of interacting spin chains governed by the Heisenberg model [1,2]. Often the reason is simple: the relevant interactions between magnetic ions are along one-dimensional chains whereas the energy exchange between the different chains is negligible [3]. Using a real-space decomposition scheme and the Trotter formula for the noncommuting operators the zero- and finite-temperature behaviors of the Heisenberg chains can be studied.

2 The Thermodynamic Properties

We consider the $S = 1/2$ Heisenberg Hamiltonian consisting of an isotropic exchange term and the Zeeman term, where the g–factors and μ_B have been set to one. J is an exchange coupling constant, B is an external magnetic field acting along the α–th coordinate axis ($\alpha = x, y, z$) and N stands for the number of sites.

$$\mathcal{H} = -J \sum_{i=1}^{N-1} \left(S_i^x S_{i+1}^x + S_i^y S_{i+1}^y + S_i^z S_{i+1}^z \right) - B \sum_{i=1}^{N} S_i^\alpha . \tag{1}$$

For the spin system described in (1) we can calculate the canonical partition function $\mathcal{Z} = \mathbf{Tr}\, e^{-\beta\mathcal{H}}$. The values of matrix elements of $e^{-\beta\mathcal{H}}$ cannot be calculated for large N because of noncommuting operators in (1). To eliminate this restriction, we look for systematic approximants to the partition function $\mathcal{Z}$. We express Hamiltonian (1) as a sum of the spin–pair Hamiltonians $\mathcal{H}_{i,i+1}$ and in the checkerboard decomposition we divide the Hamiltonian (1) into two noncommuting parts [4] each part defined by the commuting spin–pair operators $\mathcal{H}_{i,i+1}$. Then the series of the classical approximants of the quantum thermal values can be found, using the general Suzuki–Trotter formula [4]. The partition function is calculated from the expression

$$\mathcal{Z} = \lim_{m\to\infty} \mathcal{Z}_m = \lim_{m\to\infty} \mathbf{Tr} \left[\prod_{i=1}^{N/2} \mathcal{V}_{2i-1,2i} \prod_{i=1}^{N/2} \mathcal{V}_{2i,2i+1} \right]^m = \lim_{m\to\infty} \mathbf{Tr} \left[\mathcal{V}_1 \mathcal{V}_2 \right]^{mN/2} , \tag{2}$$

where $\mathcal{V}_{i,i+1} = e^{-\beta\mathcal{H}_{i,i+1}/m}$, $i = 1, 2, \cdots, N$ and m is a natural number (referred to as the Trotter number). For odd sites i all $\mathcal{V}_{i,i+1}$ become the same form $\mathcal{V}_1$, whereas for even sites the same form $\mathcal{V}_2$. So, the approximant $\mathcal{Z}_m$ can be calculated numerically, without any restrictions on the value of N, by the quantum transfer–matrix (QTM) method. In the limit $N \to \infty$ the partition function $\mathcal{Z}$ is equal to the highest eigenvalue of the global transfer matrix (TM) $\mathcal{V} = \mathcal{V}_1\mathcal{V}_2$. The computation of $\mathcal{Z}_m$ is possible for relatively small values of m, because of the size of the transfer matrix which is crucial for storage usage. As the leading errors in taking a finite m approximant are of the order of $1/m^2$, an extrapolation to infinity is reasonable but not always reliable enough.

For infinite chains (the macroscopic limit) it is better to reverse the transfer direction. For this purpose, we must define a new local transfer matrix $\mathcal{L}_{r,r+1}$ and a unitary shift operator $\mathcal{D}$ [5]. These operators act in a Hilbert space $\mathcal{H}^{2m}$ whose dimension is independent of N. In this case the global TMs can be expressed in terms of two operators $\mathcal{L}_{1,2}$ and $\mathcal{L}_{2,3}$: $\mathcal{W}_r = \left(\mathcal{L}_{r,r+1}\mathcal{D}^+\right)^m$ $(r = 1, 2)$ and the m–th classical approximant to the partition function can be written as:

$$Z_m = Tr \left(\mathcal{W}_1 \mathcal{W}_2 \right)^{N/2} . \tag{3}$$

As previously, in the limit $N \to \infty$ the partition function $\mathcal{Z}$ is equal to the highest eigenvalue of the global transfer matrix $\mathcal{W} = \mathcal{W}_1\mathcal{W}_2$. The thermodynamic functions are related to the free energy which can be calculated by means of the formula $f = -k_B T \ln Z$.

2.1 The Extrapolation Method

On the example of the specific heat $C/T = -(\partial^2 f/\partial T^2)_B$ we present how effectively extrapolate data beyond the region available for the QTM approach. To obtain a certain value of specific heat calculated from the partition function (3), the dependence of the calculated specific heat on Trotter index m should be analyzed for a given temperature and extrapolated to infinity where the leading term is known $C(1/m^2)$. For low temperatures the dependence $C(1/m^2)$ significantly deviates from linear character and with increasing m the specific heat values change significantly. It is worth mentioning that Quantum Monte Carlo methods have the advantages of linear scaling with system size and no systematic approximations, so they can also be used to study the low-temperature dependence of the specific heat [6,7].

In order to improve the accuracy of the QTM-based estimation for low temperatures, the analysis of the specific heat as a function of $1/m^2$ was made [8]. A function described by the extrapolation polynomial of the degree k ($k = 1, \ldots, k_{max}$) in the form $C_m/T = \sum_{j=0}^{k} a_j \cdot \left(\frac{1}{m^2}\right)^j$ was developed to fit all the points C_m corresponding to $m_{min}, \ldots, m_{max}$. The extrapolation procedure starts with $m_{min} = 2$ and is continued till $m = m_{max} - 1$. In each step the number of fitted points n ($n = m_{max} - m_{min} + 1$) is fixed and a number of extrapolations are performed with polynomials of the degree k ($1 \leq k \leq n-1$, but not more than 10). In this way for a given field and temperature we obtain a set of extrapolated values for different values of n and k and we can present the variation of the data with n for the fixed degree k of the polynomial. We observed that the variation of the data decreases with increasing degree k and the reliability of the estimates increases for $k \geq 4$ (see Fig. 1).

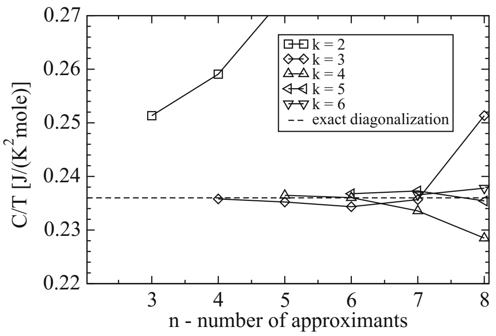


Fig. 1. The extrapolated values of specific heat versus the number of points n for which the polynomials are constructed. Each particular plot corresponds to the polynomial of a given degree k. The figure has been drawn for the finite chain with $N = 20$ where $B = 0$ and $T = 3.0\,\text{K}$. A dashed line presents results based on the exact diagonalization method. The error bars are smaller than the symbol sizes.

2.2 The DMRG Method

Due to new slow-relaxing magnetic nanosystems, a scientific interest has been focused on one-dimensional magnetic materials with large spins [9] recently. Unfortunately, since a size of the Hilbert space grows here dramatically, only calculations with $m < 10$ are available in practice. It has occurred that to increase the length of the transfer matrix in the Trotter direction, the density-matrix renormalization-group (DMRG) method [10] can be applied. The method was originally developed by White [11] for quantum spin chains at the ground state. Its adaptation - based on the transfer matrix approach - to the calculation of the thermodynamical properties of one-dimensional quantum systems was first proposed by Bursill et al. [12] and fully developed by Wang and Xiang [13] as well as by Shibata [14]. The iterative truncation algorithm for constructing the effective transfer matrices has been applied to a number of systems [15–18] for attaining the Trotter index m of several dozens.

3 Time Evolution

On the other hand, the DMRG method has occurred to be a variational method within the space of Matrix Product States (MPS) [19]. It is related to the fact that for physical systems, e.g. where only nearest-neighbours interactions are present, only minor part of Hilbert space is involved [20–22]. It corresponds to assigning a finite entanglement content to spins in the ground state. Therefore, any state of the spin chain can be presented in the MPS representation

$$|\psi\rangle = \sum_{\sigma_1,\ldots,\sigma_N}^{d_1,\ldots,d_N} \sum_{a_1,\ldots,a_{N-1}}^{D_1,\ldots,D_{N-1}} M^{\sigma_1}_{1,a_1} M^{\sigma_2}_{a_1,a_2} \ldots M^{\sigma_{N-1}}_{a_{N-2},a_{N-1}} M^{\sigma_N}_{a_{N-1},1} |\sigma_1 \ldots \sigma_N\rangle$$

where d_i is dimension of the local base $\{\sigma_i\}$ at the i–th site whereas D_i are related to the entanglement of neighbouring spins. In an analogous manner any operator can be written as a Matrix Product Operator (MPO):

$$\mathcal{O} = \sum_{\sigma_1,\ldots,\sigma_N}^{d_1,\ldots,d_N} \sum_{\sigma'_1,\ldots,\sigma'_N}^{d_1,\ldots,d_N} W^{\sigma_1\sigma'_1} W^{\sigma_2\sigma'_2} \ldots W^{\sigma_N\sigma'_N} |\sigma_1 \ldots \sigma_N\rangle\langle\sigma'_1 \ldots \sigma'_N|$$

Due to the above representation the state space grows only polynomially in the system size (not exponentially as usual). Thus, the time of calculations is significantly reduced for one-dimensional strongly correlated systems, taking into account both their static and dynamic properties.

When the variational principle is applied, the ground state can be found very smoothly by the minimization procedure $\langle\psi|\mathcal{H}|\psi\rangle$ under the constrain $\langle\psi|\psi\rangle = 1$ [23]. Moreover, the time evolution can also be performed very efficiently. So, discretize time as $t = n\Delta t$ can be used and when for the Hamiltonian (1) a second-order Trotter decomposition is applied [22] the time-evolution operator can be presented as:

$$e^{-i\mathcal{H}\Delta t} = e^{-i\mathcal{H}_o \Delta t/2} e^{-i\mathcal{H}_e \Delta t} e^{-i\mathcal{H}_o \Delta t/2} + O(\Delta t^3)$$

where

$$\mathcal{H}_o = -J \sum_{i=1}^{N/2} \left(S^x_{2i-1} S^x_{2i} + S^y_{2i-1} S^y_{2i} + S^z_{2i-1} S^z_{2i} \right)$$

$$\mathcal{H}_e = -J \sum_{i=1}^{N/2-1} \left(S^x_{2i} S^x_{2i+1} + S^y_{2i} S^y_{2i+1} + S^z_{2i} S^z_{2i+1} \right) - B \sum_{i=1}^{N} S^z_i$$

Then the time evolution algorithm takes a very simple form [22]: one starts from $|\psi_0\rangle$ and repeats the following steps

1. Applying the MPO of the odd bonds to $|\psi(t)\rangle$.
2. Applying the MPO of the even bonds to $e^{-i\mathcal{H}_o \Delta t/2}|\psi(t)\rangle$.
3. Compressing the MPS $|\psi(t+\Delta t)\rangle = e^{-i\mathcal{H}_e \Delta t/2} e^{-i\mathcal{H}_o \Delta t/2}|\psi(t)\rangle$ to the starting dimension.

It is worth adding that finite temperature calculations can be carried out based on mixed-states time evolution [22].

4 Simulations Results

Molecular materials have already shown their great potential [2]. When magnetic properties are considered, one of the examples are the ferromagnetic bimetallic and polymetallic complexes that stimulated interest in the quantum chain Heisenberg model applied to investigate various mixed spin systems. For the thiocyanate bridge compounds the anisotropic alternating ($S = 1/2$ and $S = 3/2$) spin Hamiltonian [24] is employed. Its formula is such as in (1), but the Pauli operators located on even sites have been replaced by 3/2 spin operators. The high accuracy results are successfully fitted to the corresponding experimental susceptibility and magnetization data measured [24].

4.1 Finite-Temperature Static Properties

In this section we present numerical results for the infinite chain. The main numerical problem is the calculation of the global transfer matrix $\mathcal{W}$ which is given as a complex sum of nested multiplications of the matrix elements $\mathcal{V}_{i,i+1}$ (2). The calculation of the original transfer matrix elements was distributed among a number of processes. We have used coarse-grained homoparallelism: the work was split on the identical independent subtasks [25].

We have tested the parallel version of our algorithm on the part of the full code, where individual elements of the transfer matrix are the subject of distribution between the parallel processes. It has been developed using the MPI

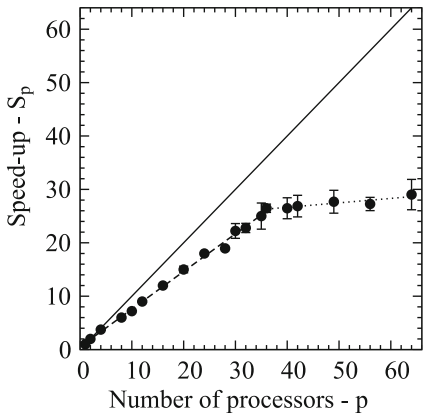


Fig. 2. The speed-up S_p of the parallel computation for the number of processors $p \leq 64$. The full line describes the function $S_p = p$. For $p < 30$ the error bars are smaller than the symbol size. The dashed line corresponds to the slope 0.75, whereas the dotted one to an almost horizontal line (the lines are only a guide for the eye).

library and implemented on Intel Core microarchitecture (IA-32) with 46 dual-core Xeon EM64T 3 GHz processors. We have analyzed the speed-up which has been computed as the quotient of the CPU time T_s of sequential version of the algorithm divided by the maximum CPU time T_p used by parallel algorithm [26]. For the sequential version of the algorithm we need about 1800 s of CPU time.

The speed-up of the parallel job is drawn in Fig. 2. One can see that when the number of processors p is less than 35, an efficiency is close to 75 %. Further increasing of p is practically useless as dots form here almost a horizontal line. We suppose that the complexity of the calculations (e.g., the size of the transfer matrix) is a reason. We expect that for the full DMRG code (more complex) the better scalability would be found.

4.2 Zero-Temperature Dynamic Properties

Our initial attempts to parallelize the code using the MPS formalism, have not been successful. However, it was found that the evolution operator acting on the state can run in parallel on each site. The neck of the bottle is compressing the MPS states to the starting dimension, which must occur sequentially, site by site.

In order to study the ground-state magnetization dynamics we have considered the alternating Heisenberg chain with $N = 100$ sites ($S = 3/2$ on odd sites and $s = 1/2$ on even sites). To perform the time evolution of the magnetization, first the magnetic field B was fixed as $B^x = 0.5$ (for the case of a weak field) or $B^x = 5$ (for the case of a strong field) determining the initial state $|\psi_0\rangle$ which is calculated by a variational optimisation scheme. Next, at time $t = 0$ the field B is changed instantaneously to the value $B^z = 0.5$ and the unitary time evolution

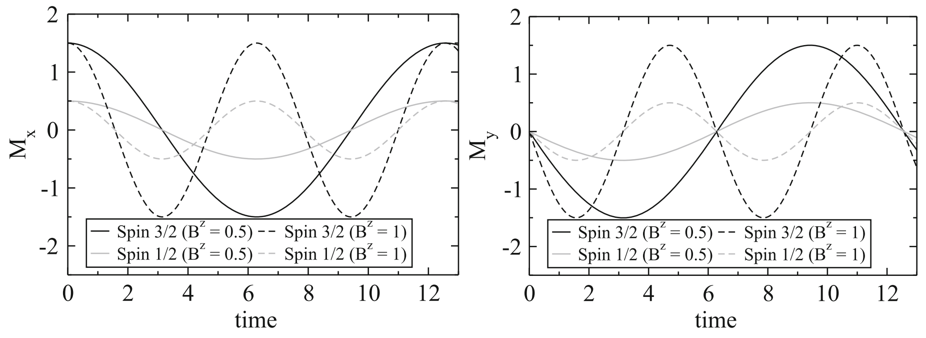


Fig. 3. The time evolution of the magnetization components for the ferromagnetic alternating chain. The calculations were done for the two central spins of the chain. The z-th component is zero with the uncertainty smaller than 10^{-9}.

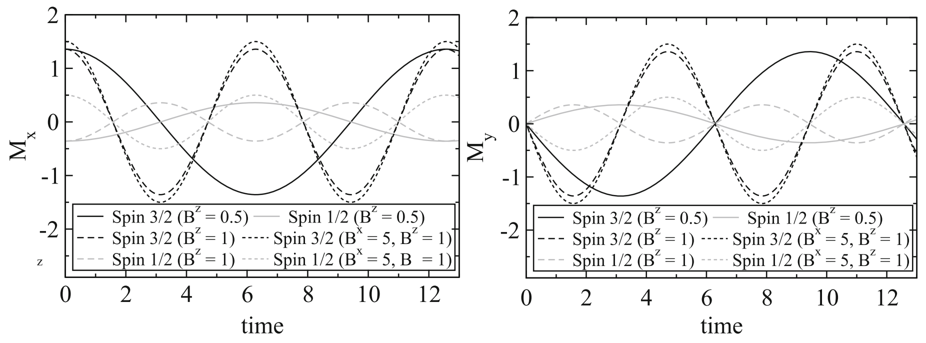


Fig. 4. The time evolution of the magnetization components for the antiferromagnetic alternating chain. The calculations were performed for the two central spins. The z-th component is zero with the uncertainty smaller than 10^{-9}.

of the system is performed. The initial state $|\psi_0\rangle$ evolves and the magnetization is recorded for subsequent moments of time. Our results are presented in Figs. 3 and 4 for the ferromagnetic $(J > 0)$ and antiferromagnetic $(J < 0)$ cases, respectively.

The curves in Fig. 3 demonstrate the influence of the magnetic field B^z for the period of oscillation of the magnetization components M^x and M^y. In order to understand the frequency of the oscillations we have studied a single spin $S = 1/2$ experiencing only Zeeman interaction with an external magnetic field B^z that can be solved analytically. We have found that the M^x (M^y) magnetization varies with time as a cosine function $\cos(B^z t)$ $(-\sin(B^z t))$. Because the total magnetisation operator commutes with the Hamiltonian the M^z is zero all time.

For the antiferromagnetic case (Fig. 4), of course, the same relation between the period of the oscillations and the value of the field is observed. A significant difference is the behaviour of the amplitudes of the oscillations. While for the ferromagnetic case the amplitude of magnetization reaches the value 3/2 or 1/2

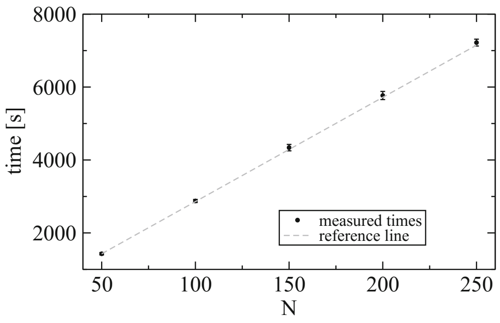


Fig. 5. Time of computations for the alternating Heisenberg chain with various lengths at the initial field $B^x = 0.5$. The D_i parameters are equal to 16.

depending on the spin, for the antiferromagnetic case the magnetization amplitude is always somewhat smaller. Moreover, for the antiferromagnetic case the strong initial magnetic field B^x changes substantially the initial state setting all the spins in accordance with the field. As a result, both magnetizations oscillate as in the ferromagnetic case.

Finally, we have decided to evaluate the time complexity of MPS-based calculations for the ground-state magnetization dynamics. For each of lengths $N = 50, 100, 150, 200, 250$ several computations have been performed for the weak initial field. As one can see in Fig. 5 the average time grows linearly with a chain length. Since in the direct computations the time grows exponentially the MPS-based calculations provide a huge increase in speed. Moreover, the memory size is also reduced dramatically: the number of elements of the state vector is decreased from $8^{\frac{N}{2}}$ to $192N$ while the number of Hamiltonian elements is scaled down from 8^N to 600.

There are two sources of error: the noncommutativity of Hamiltonian operators when the Trotter decomposition is applied and truncation of the bond dimensions of the MPS after each time step. As a measure of the error, one can take the energy that is fixed during the time evolution. It turned out that ferro- and antiferromagnetic systems are subject to different energy errors. While for the former, the error is around 10^{-12}, then for the antiferromagnetic case it is five orders larger (energy values are of the order 10). Although the intermediate area of the initial field between $B^x = 0.5$ and $B^x = 5$ has not been systematically studied, we noticed that there is an error even greater.

5 Conclusions

Our results have shown that for the $S = 1/2$ spin systems one can go to relatively large m and, providing an appropriate extrapolation, get very accurate results for thermodynamic functions. The accuracy of the present QTM estimated results is comparable with that of DMRG results for temperature $T = 2\,\mathrm{K}$, $(k_B T/J = 0.077)$ differing by less than $1\,\%$.

For systems with higher spins, because of the large size of matrices, one is able to go only to $m < 10$ and even providing very efficient extrapolation technique the accurate results are beyond our reach. Then the application of the DMRG approach to the QTM method seems to be promising [24] allowing the construction of effective transfer matrices with high m. In order to accelerate our calculations, we have tested the parallel version of the code, where the original transfer matrix is build. We have found that above a certain number of processors the speed-up grows extremely slowly. In our opinion it is related to the low complexity of the calculations. We expect that it is possible to estimate the optimal number of processors with respect to the code complexity, but it requires further considerations.

In order to simulate the dynamics of quantum chains the Matrix Product States representation was employed. The time evolution of the zero-temperature magnetization has been performed after the sudden change in applied field. Our results show significantly different behavior depending on the initial value of the magnetic field. We have not built the parallel version of the code but the time complexity of MPS-based calculations has occurred to be linear.

Acknowledgments. The calculations were performed on computer facilities granted by Poznan Supercomputing and Networking Centre (Poland) as well as within DECI programme by the PRACE-2IP (FP7/2007-2013) under grant agreement no RI-283493. Support from the Polish MNiSW through the grant No N519 579138 is also acknowledged.

References

1. Kahn, O.: Molecular Magnetism. Wiley-VCH, New York (1993)
2. Gatteschi, D., Sessoli, R., Villain, J.: Molecular Nanomagnets. Oxford University Press, Oxford (2006)
3. Steiner, M., Villain, J., Windsor, C.G.: Theoretical and experimental studies on one-dimensional magnetic systems. Adv. Phys. **25**, 87 (1976)
4. Delica, T., Leschke, H.: Formulation and numerical results of the transfer-matrix method for quantum spin chains. Physica A **176**, 736 (1990)
5. Kamieniarz, G., Matysiak, R.: Transfer matrix simulation technique: effectiveness and applicability to the low-dimensional magnetic spin systems. J. Comput. Appl. Math. **189**, 471 (2006)
6. Syljuåsen, O.F., Sandvik, A.W.: Quantum Monte Carlo with directed loops. Phys. Rev. E **66**, 046701 (2002)
7. Androvitsaneas, P., Fytas, N.G., Paspalakis, E., Terzis, A.F.: Quantum Monte Carlo simulations revisited: the case of anisotropic Heisenberg chains. Philos. Mag. **92**, 4649 (2012)
8. Matysiak, R., Kamieniarz, G., Gegenwart, P., Ochiai, A.: Specific heat of the polydomain Yb_4As_3 system: agreement between spin - 1/2 modelling and experiment. Phys. Rev. B **79**, 224413 (2009)
9. Coulon, C., Miyasaka, H., Clerac, R.: Single-chain magnets: theoretical approach and experimental systems. Struct. Bond. **122**, 163 (2006)

10. Schollwoeck, U.: The density-matrix renormalization group. Rev. Mod. Phys. **77**, 259 (2005)
11. White, S.R.: Density-matrix algorithms for quantum renormalization groups. Phys. Rev. B **48**, 10345 (1993)
12. Bursill, R.J., Xiang, T., Gehring, G.A.: The density matrix renormalization group for a quantum spin chain at non-zero temperature. J. Phys. Condens. Matter **8**, L583 (1996)
13. Wang, X.Q., Xiang, T.: Transfer-matrix density-matrix renormalization-group theory for thermodynamics of one-dimensional quantum systems. Phys. Rev. B **56**, 5061 (1997)
14. Shibata, N.: Thermodynamics of the anisotropic Heisenberg chain calculated by the density matrix renormalization group method. J. Phys. Soc. Jpn. **66**, 2221 (1997)
15. Sobczak, P., Barasinski, A., Drzewinski, A., Kamieniarz, G., Klak, J., Bienko, A., Mrozinski, J.: Magnetic properties and DMRG modeling of the 1D bimetallic thiocyanate bridged compound $(CuL_1)[Co(NCS)_4(L_1 = N - rac - 5, 12 - Me_2 - [14] - 4, 11 - dieneN_4)$. Polyhedron **28**, 1838 (2009)
16. Barasinski, A., Drzewinski, A., Kamieniarz, G.: Quantum effects and Haldane gap in magnetic chains with alternating anisotropy axes. Comput. Phys. Commun. **182**, 2013 (2011)
17. Sobczak, P., Barasinski, A., Kamieniarz, G., Drzewinski, A.: Anisotropic planar Heisenberg model of the quantum heterobimetallic zigzag chains with bridged $Re^{IV} - Cu^{II}$ magnetic complexes. Phys. Rev. B **84**, 224431 (2011)
18. Barasinski, A., Kamieniarz, G., Drzewinski, A.: Magnetization-based assessment of correlation energy in canted single-chain magnets. Phys. Rev. B **86**, 214412 (2012)
19. Östlund, S., Rommer, S.: Thermodynamic limit of density matrix renormalization. Phys. Rev. Lett. **75**, 3537 (1995)
20. Verstraete, F., Cirac, K.: Matrix product states represent ground states faithfully. Phys. Rev. B **73**, 094423 (2006)
21. Verstraete, F., Murg, V., Cirac, K.: Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems. Adv. Phys. **57**, 143 (2008)
22. Schollwoeck, U.: The density-matrix renormalization group in the age of matrix product states. Ann. Phys. **326**, 96 (2011)
23. Wozniak, D., Drzewinski, A., Kamieniarz, G.: Matrix-product states for the Ising model in a transverse field. Acta Phys. Superficierum **12**, 187 (2012)
24. Barasinski, A., Sobczak, P., Drzewinski, A., Kamieniarz, G., Bienko, A., Mrozinski, J., Gatteschi, D.: Anisotropy and magnetic properties of the bimetallic thiocyanate-bridged chains: density-matrix renormalization approach. Polyhedron **29**, 1485 (2010)
25. Bauer, Barr E.: Practical Parallel Programming. Academic Press Inc, San Diego (1992)
26. Van de Velde, E.F.: Concurrent Scientific Computing. Springer, New York (1994)

Minisymposium on Applied High Performance Numerical Algorithms in PDEs

A Domain Decomposition Method for Discretization of Multiscale Elliptic Problems by Discontinuous Galerkin Method

Maksymilian Dryja(✉)

Department of Mathematics, Univeristy of Warsaw,
Banacha 2, 02-097 Warsaw, Poland
dryja@mimuw.edu.pl

Abstract. In this paper boundary value problems for second order elliptic equations with highly discontinuous coefficients are considered on a 2D polygonal region. The problems are discretized by a discontinuous Galerkin (DG) with finite element method (FEM) on triangular elements using piecewise linear functions.

The goal is to design and analyze a parallel algorithm for solving the discrete problem whose rate of convergence is independent of the jumps of the coefficients. The method discussed is an additive Schwarz method (ASM) which belongs to a class of domain decomposition methods and is one of the most efficient parallel algorithm for solving discretizations of PDEs.

It turns out that the convergence of the method presented here is almost optimal and only weakly depends on the jumps of coefficients. The suggested method is very well suited for parallel computations.

Keywords: Interior penalty method · Discontinuous Galerkin method · Elliptic equations with discontinuous coefficients · Finite element method · Additive Schwarz method

1 Introduction

We consider boundary value problems (BVPs) for second order elliptic equations with highly discontinuous coefficients posed on a 2D polygonal region. The problem is discretized by a discontinuous Galerkin (DG) method with FEM on triangular elements and piecewise linear functions, see [1,3], and references therein. The goal of this paper is to design and analyze a parallel algorithm for solving the discrete problem with rate of convergence independent of the jumps of coefficients.

The proposed algorithm is an additive Schwarz method (ASM) with overlaps and belongs to a class of domain decomposition methods and it is one of the most efficient parallel algorithms for solving discretizations of PDEs, see [5].

This work was supported in part by the Polish National Science Centre Grant no 2011/01/B/ST1/01179.

In the paper the results obtained in [4] for continuous piecewise linear finite element discretization are extended to DG discretization. They are more general comparising to the results of [4].

The presented ASM is two-level with a special coarse space defined on large triangles of coarse triangulation, i.e. a multiscale coarse space. This is a space of continuous functions which are discrete harmonic on edges of the coarse triangles and inside of them in the sense of corresponding bilinear forms. The local spaces are defined in a standard way, on the fine triangulation, on extensions of coarse triangles; these spaces contain discontinuous functions. For literature on the topic see [4,5], and references therein.

It turns out that the convergence of the discussed ASM is dependent of the jumps of the coefficients on the boundary of coarse triangles only. For some distributions of jumps, the convergence of the ASM is also independent of these jumps.

The paper is organized as follows. In Sect. 2, differential and discrete problems are formulated. In Sect. 3, a two level ASM for solving the discrete problem is designed and analyzed. The main result is Theorem 5, which guarantees the optimality of the method. Section 4 is devoted to an implementation of the method discussed.

2 Differential and Discrete DG Problems

We consider the following elliptic problem:
Find $u^* \in H_0^1(\Omega)$ such that

$$a(u^*, v) = f(v), \qquad \forall v \in H_0^1(\Omega) \tag{1}$$

where

$$a(u,v) = \int_\Omega \rho(x) \nabla u \cdot \nabla v dx, \qquad f(v) = \int_\Omega f v dx.$$

We assume that Ω is a polygonal region, $f \in L^2(\Omega)$ and $\rho(x) \geq \rho_0 > 0$, and $\rho \in L^\infty(\Omega)$. Under these assumptions problem (1) is well posed.

We will also assume that $\rho_0 \geq 1$. This can be fulfilled by scaling (1). It is used in the analysis of preconditioner discussed in Sect. 3.

Let $\mathcal{T}^h(\Omega)$ be a triangulation of Ω with triangular elements K_i and the mesh parameter h. It is constructed as a refinement of the coarse triangulation of Ω consisting of large triangles Ω_l of diameter $H_l, l = 1, \cdots, L, H_l = diam(\Omega_l)$ and $H = \max H_l$. The refinement procedure is repeated several times, where one step of the process is to split each triangle into four smaller ones, obtained by connecting the midpoints of its edges. Let $X_i(K_i)$ denote a space of linear functions on K_i and

$$X_h(\Omega) = \Pi_{i=1}^N X_i(K_i), \qquad \bar{\Omega} = \cup_{i=1}^N K_i,$$

be the space in which problem (1) is approximated. Note that $X_h(\Omega) \not\subset H^1(\Omega)$ and its elements do not vanish on $\partial\Omega$, in general.

The discrete problem for (1) is of the form:
Find $u_h^* \in X_h(\Omega)$ such that

$$\hat{a}_h(u_h^*, v_h) = f(v_h), \qquad v_h \in X_h(\Omega), \tag{2}$$

where for $u, v \in X_h(\Omega), u = \{u_i\}_{i=1}^N, u_i \in X_i(K_i)$,

$$\hat{a}_h(u,v) = \sum_{i=1}^N \hat{a}_i(u,v), \qquad f(v) = \sum_{i=1}^N \int_{K_i} f v_i dx.$$

Since we use linear elements, we can assume without loss of generality that $\rho_{|K_i} = \rho_i$ is constant on K_i. Here

$$\hat{a}_i(u,v) = a_i(u,v) + s_i(u,v) + p_i(u,v),$$

$$a_i(u,v) = \int_{K_i} \rho_i \nabla u_i \cdot \nabla v_i dx,$$

$$s_i(u,v) = \sum_{E_{ij} \subset \partial K_i} \int_{E_{ij}} \omega_{ij} [n_i^T \rho_i \nabla u_i (v_j - v_i) + n_i^T \rho_i \nabla v_i (u_j - u_i)]\, ds,$$

$$p_i(u,v) = \sum_{E_{ij} \subset \partial K_i} \frac{\sigma}{h} \int_{E_{ij}} \gamma_{ij} (u_i - u_j)(v_i - v_j)\, ds$$

where $E_{ij} = E_{ji} = \partial K_i \cap \partial K_j, E_{ij} \subset \partial K_i$ and $E_{ji} \subset \partial K_j$; $n_i = n_{E_{ij}}$ is the unit normal vector to E_{ij} pointing from K_i to K_j;

$$\omega_{ij} \equiv \omega_{E_{ij}} = \frac{\rho_j}{\rho_i + \rho_j}, \qquad \omega_{ji} \equiv \omega_{E_{ji}} = \frac{\rho_i}{\rho_i + \rho_j}$$

and

$$\gamma_{ij} \equiv \gamma_{E_{ij}} = \frac{2\rho_i \rho_j}{\rho_i + \rho_j};$$

σ is a positive penalty parameter (sufficiently large, see below Lemma 1). For boundary egdes these definitions extend straightforwardly, setting for $E_{ij} \subset \partial\Omega$: $\omega_{ij} = 1$, $\omega_{ji} = 0$, $v_j = u_j = 0$ and $\gamma_{ij} = \rho_i$.

To analyze problem (2) we introduce some auxiliary bilinear forms and a broken norm. Let

$$d_h(u,v) = \sum_{i=1}^N d_i(u,v), \quad d_i(u,v) = a_i(u,v) + p_i(u,v) \tag{3}$$

and let the weighted broken norm in $X_h(\Omega)$ be defined by

$$\| u \|_{1,h}^2 \equiv d_h(u,u) = \sum_{i=1}^N \{ \| (\rho_i)^{1/2} \nabla u_i \|_{L^2(K_i)}^2 + \sum_{E_{ij} \subset \partial K_i} \frac{\sigma}{h} \gamma_{ij} \| u_i - u_j \|_{L^2(E_{ij})}^2 \}. \tag{4}$$

Lemma 1. *There exists $\sigma_0 > 0$ such that for $\sigma \geq \sigma_0$ there exist positive constants C_0 and C_1 independent of ρ_i and h such that for any $u \in X_h$ hold*

$$C_0 d_i(u,u) \leq \hat{a}_i(u,u) \leq C_1 d_i(u,u)$$

and

$$C_0 d_h(u,u) \leq \hat{a}(u,u) \leq C_1 d_h(u,u).$$

For the proof we refer the reader to [2]; see also [3] or [1].

Lemma 1 implies that the discrete problem (2) is well posed if the penalty parameter $\sigma \geq \sigma_0$. Below σ is fixed and assumed to satisfy the above condition.

The error bound is given by

Theorem 2. *Let u^* and u_h^* be the solutions of (1) and (2). For $u^*_{|K_i} \in H^2(K_i)$ holds*

$$\| u^* - u_h^* \|_{1,h}^2 \leq M h^2 \sum_{i=1}^{N} \rho_i |u^*|^2_{H^2(K_i)}$$

where M is independent of h, u^* and ρ_i.

The proof follows from Lemma 1; for details see, for example, [3].

3 ASM with a Multiscale Coarse Space

We design and analyze a two-level additive Schwarz method (ASM) for solving the discrete problem (2). For that the general theory of ASMs is used, see [5]. The decomposition of $X_h(\Omega)$ consists of the local spaces defined on subdomains extended from the coarse triangles Ω_l, and the global space of continuous discrete harmonic functions related to the coarse triangulation.

3.1 Decomposition of $X_h(\Omega)$

Let

$$X_h(\Omega) = V^{(0)}(\Omega) + \sum_{l=1}^{L} V^{(l)}(\Omega) \tag{5}$$

where $V^{(0)}(\Omega)$ is a coarse space while $V^{(l)}(\Omega), l = 1, \ldots L$, are local spaces associated with Ω_l. They are defined as follows. For $l = 1, \ldots, L$, Ω_l is extended to Ω_l' by adding triangles from the fine triangulation around $\partial\Omega_l$ which intersect $\partial\Omega_i$ by vertex and/or edge. In this way we get an overlapping partitioning of Ω,

$$\bar{\Omega} = \bigcup_{l=1}^{L} \bar{\Omega}_l'$$

with overlap $\delta_l \approx 2h$ defined as

$$\delta_l = dist(\partial\Omega_l' \setminus \partial\Omega, \partial \overset{o}{\Omega}_l \setminus \partial\Omega)$$

where $\overset{o}{\Omega}_l$ denotes the interior part of Ω_l which is not overlapped by any other Ω_p for $p \neq l$; see [5, p. 198] for figures which exemplify such decomposition.

The local spaces $V^{(l)}(\Omega)$ for $l = 1, \ldots, L$ are defined as

$$V^{(l)}(\Omega) = \{\{v_i\}_{i=1}^N \in X_h(\Omega) : v_i = 0 \text{ on } K_i \not\subset \bar{\Omega}_l'\}. \tag{6}$$

Thus $V^{(l)}(\Omega)$ is the restriction of $X_h(\Omega)$ to $\bar{\Omega}_l'$ and zero outside of $\bar{\Omega}_l'$.

The coarse space $V^{(0)}(\Omega)$ is defined in a special way. The functions in $V^{(0)}(\Omega)$ are going to be piecewise linear continuous on the fine triangulation and discrete harmonic on $\partial\Omega_l$ and in Ω_l. Let ν be the set of all vertices of $\bar{\Omega}_l$. With each $x^{(k)} \in \nu$, a function $\Phi_k(x)$ is associated with support on a union of coarse triangles Ω_l for which $x^{(k)}$ is a common vertex. On the set ν, we set $\Phi_k(x^{(k)}) = 1$ and $\Phi_k(x) = 0$ otherwise. Next we define Φ_k on the boundary of each Ω_l. Let $x^{(k)}$ be a vertex of Ω_l and let F_{lp} denote an edge of Ω_l shared with Ω_p, $F_{lp} = \partial\Omega_l \cap \partial\Omega_p$. Let $a_{\Omega_l}(\cdot,\cdot)$ be the restriction of $a(\cdot,\cdot)$ to $\bar{\Omega}_l$, i.e.

$$a_{\Omega_l}(u,v) = \sum_{K_i \subset \bar{\Omega}_l} (\rho_i \nabla u, \nabla v)_{L^2(K_i)} = (\rho^{(l)} \nabla u, \nabla v)_{L^2(\Omega_l)}, \tag{7}$$

where by definition $\rho^{(l)} = \rho_i$ on $K_i \subset \bar{\Omega}_l$. Then the restriction of $a_{\Omega_l}(\cdot,\cdot)$ to F_{lp} is defined as

$$a_{F_{lp}}(u,v) = (\rho_{lp} D_\tau u, D_\tau v)_{L^2(F_{lp})} \tag{8}$$

where D_τ is the tangential derivative and ρ_{lp}, on F_{lp}, is the harmonic average of the coefficients on Ω_l and Ω_p, i.e. $\rho_{lp} = 2\rho^{(l)}\rho^{(p)}/(\rho^{(l)} + \rho^{(p)})$. Note that in this way $\rho_{lp} = \rho_{pl}$. On F_{lp}, we define the values of Φ_k as the solution of the one-dimensional problem:

$$a_{F_{lp}}(\Phi_k, v) = 0 \qquad \forall v \in \overset{o}{V}_h(F_{lp}) \tag{9}$$

with Dirichlet boundary conditions $\Phi_k(x^{(k)}) = 1$ and $\Phi_k(x^{(m)}) = 0$ at the other end, $x^{(m)}$, of F_{lp}. Above, $\overset{o}{V}_h(F_{lp})$ is the set of piecewise linear continuous functions with zero values on ∂F_{lp}. In addition we set $\Phi_k(x) = 0$ on those edges of Ω_l which do not end at $x^{(k)}$.

Finally we extend Φ_k, already defined on $\partial\Omega_l$, into Ω_l as a discrete harmonic function in the sense of $a_{\Omega_l}(\cdot,\cdot)$, i.e.

$$\begin{cases} a_{\Omega_l}(\Phi_k, v) = 0, \qquad \forall v \in \overset{o}{V}_h(\Omega_l) \\ \text{with } \Phi_k(x) \text{ on } \partial\Omega_l \text{ defined in (9).} \end{cases} \tag{10}$$

Here $v \in \overset{o}{V}_h(\Omega_l)$ is a set of piecewise linear continuous functions defined on $\bar{\Omega}_l$ with zero values on $\partial\Omega_l$.

Using these functions, the coarse space $V^{(0)}(\Omega)$ is defined as

$$V^{(0)} = span\{\Phi_k(x)\}_{x^{(k)} \in \nu}. \tag{11}$$

Of course $V^{(0)} \subset X_h(\Omega)$.

Remark 3. This space is called a multiscale coarse space and at the beginning it was used to obtain more accurate approximation. In [4], $V^{(0)}(\Omega)$ was used also as a coarse space in ASM for the conforming (continuous) finite element method in the case when the coefficients are piecewise constant across $\partial\Omega_l$, $l = 1, \ldots, L$.

3.2 Inexact Solver

For $u^{(0)}, v^{(0)} \in V^{(0)}(\Omega)$, let

$$b_0(u^{(0)}, v^{(0)}) = d_h(u^{(0)}, v^{(0)}), \tag{12}$$

where $d_h(\cdot,\cdot)$ is defined in (3).

For $l = 1, \ldots, L$ we set

$$b_l(u^{(l)}, v^{(l)}) = d_{\Omega_l'}(u^{(l)}, v^{(l)}), \quad u^{(l)}, v^{(l)} \in V^{(l)}(\Omega) \tag{13}$$

where for $u^{(l)} = \{u_i^{(l)}\}_{i=1}^N, \;\; v^{(l)} = \{v_i^{(l)}\}_{i=1}^N$

$$\begin{aligned} d_{\Omega_l'}(u^{(l)}, v^{(l)}) = \sum_{K_i \subset \bar{\Omega}_l'} &\{(\rho_i \nabla u_i^{(l)}, \nabla v_i^{(l)})_{L^2(K_i)} + \\ &+ \sum_{E_{ij} \subset \partial K_i} \gamma_{ij} \frac{\sigma}{h} (u_j^{(l)} - u_i^{(l)}, v_j^{(l)} - v_i^{(l)})_{L^2(E_{ij})}\}. \end{aligned} \tag{14}$$

3.3 The Operator Equation

For $l = 0, 1, \ldots, L$ let $T_l : X_h(\Omega) \to V^{(l)}(\Omega)$ be defined by

$$b_l(T_l u, v) = \hat{a}_h(u, v), \qquad v \in V^{(l)}(\Omega). \tag{15}$$

Note that $T_l u$ is defined uniquely for given $u \in X_h(\Omega)$ as the solution of local problems defined on Ω_l' for $l = 1, \ldots, L$, and the global one for $l = 0$.
Let

$$T = T_0 + T_1 + \cdots + T_L. \tag{16}$$

We replace (2) by the following operator equation

$$T u_h^* = g_h \tag{17}$$

where $g_h = \sum_{l=0}^L g_l, g_l \equiv T_l u_h^*$. Note that to compute g_l we do not need to know u_h^*, the solution of (1).

Problems (2) and (17) are equivalent, what follows from the theorem below.

To formulate the convergence theorem for the discussed ASM we have to introduce some notation. Let us for each Ω_l define

$$\bar{\rho}_l = \sup_{K_i \subset \Omega_l^h} \rho_i \tag{18}$$

where Ω_l^h is a union of triangles $K_i \subset \bar{\Omega}_l$ which intersect $\partial\Omega_l$ by vertex and/or edge.

Theorem 4. *The operator T defined in (16) satisfies $T = T^* > 0$. Moreover, for any $u \in X_h(\Omega)$ there holds*

$$C_0\beta^{-1}\hat{a}_h(u,u) \le \hat{a}_h(Tu,u) \le C_1\hat{a}_h(u,u) \tag{19}$$

where

$$\beta = \max_{l=1,\dots,L} \bar{\rho}_l \frac{H_l}{h}\left(1 + \log\frac{H_l}{h}\right)^2, \tag{20}$$

with $\bar{\rho}_l$ defined in (18), and C_0 and C_1 are positive constants independent of H, h and the jumps of $\rho(x)$.

Remark 5. The proof of Theorem 5 needs to check three key assumptions of abstract theory of ASMs, see for example the book [5]. For that we need several auxiliary lemmas, some of them are new. The proof is omitted here due to the limit of pages. It will be published elsewhere together with supporting numerical tests.

4 Implementation

Equation (17) can be solved efficiently by the conjugate gradient method. To simplify the presentation we discuss here the Richardson's method instead. The latter is of the form: given $u^{(0)}$, iterate for $n = 0, 1, \dots$

$$u^{(n+1)} = u^{(n)} - \tau^*(Tu^{(n)} - g_h) \tag{21}$$

where we can set $\tau^* = 2/(C_1 + C_0\beta^{-1})$ according to Theorem 5. Since

$$r^{(n)} \equiv (Tu^{(n)} - g_h) = \sum_{l=0}^{L}(T_l u^{(n)} - g_h) = \sum_{l=0}^{L} T_l(u^{(n)} - u_h^*) \equiv \sum_{l=0}^{L} r_l^{(n)}, \tag{22}$$

we need to compute $r_l^{(n)} \equiv T_l(u^{(n)} - u_h^*)$ for $l = 0, 1, \dots, L$. Note that these problems, see (15), are independent of each other, therefore, they can be solved in parallel. Problems for $l = 1, \dots, L$ are local, and they are defined on Ω_l'.

The problem for $l = 0$ is global and it is defined on the coarse triangulation with piecewise linear continuous functions. The solution of the coarse problem requires finding the coarse basis functions $\{\Phi_k\}$ for all the vertices $x^{(k)}$ of the coarse triangles. This is a precomputation step and it should be carried out before starting the iterative process (21).

The above implementation shows that the proposed algorithm is very well suited for parallel computations.

References

1. Arnold, D.N., Brezzi, F., Cockburn, B., Marini, L.D.: Unified analysis of discontinuous Galerkin methods for elliptic problems. SIAM J. Numer. Anal. **39**, 1749–1770 (2001)
2. Dryja, M.: On discontinuous Galerkin methods for elliptic problems with discontinuous coefficients. Comput. Methods Appl. Math. **3**, 76–85 (2003)
3. Ern, A., Stephansen, A.F., Zunino, P.: A discontinuous Galerkin method with weighted averages for advection-diffusion equations with locally small and anisotropic diffusivity. IMAJ. Numer. Anal. **29**, 235–256 (2009)
4. Graham, I.G., Lechner, P.O., Scheichl, R.: Domain decomposition for multiscale PDEs. Numer. Math. **106**(4), 589–626 (2007)
5. Toselli, A., Widlund, O.: Domain Decomposition Methods-Algorithms and Theory, Springer Series in Computational Mathematics, vol. 34. Spinger, Berlin (2005)

Parallel Preconditioner for the Finite Volume Element Discretization of Elliptic Problems

Leszek Marcinkowski[1(✉)] and Talal Rahman[2]

[1] Faculty of Mathematics, University of Warsaw, Banacha 2, 02-097 Warszawa, Poland
L.Marcinkowski@mimuw.edu.pl

[2] Faculty of Engineering, Bergen University College, Nygårdsgaten 112, 5020 Bergen, Norway
Talal.Rahman@hib.no

Abstract. In this paper we present a parallel preconditioner for the standard Finite Volume (FV) discretization of elliptic problems, using the standard continuous piecewise linear Finite Element (FE) function space. The proposed preconditioner is constructed using an abstract framework of the Additive Schwarz Method, and is fully parallel. The convergence rate of the Generalized Minimal Residual (GMRES) method with this preconditioner is shown to be almost optimal, i.e., it depends poly-logarithmically on the mesh sizes.

Keywords: Finite Volume Element · Parallel Preconditioner · Domain Decomposition Method · Additive Schwarz Method

1 Introduction

Finite Volume (FV) methods form an important class of discretization methods for solving Partial Differential Equations (PDEs), which are quite popular and have been in use in the engineering community for many years now. Finite Volume methods, in general, have some conservation properties that are desirable in many engineering applications. A Finite Volume Element (FVE) method is a FV method where a finite element space on a primal mesh is used to approximate the solution and the equation is discretized on the corresponding dual mesh, cf. [1] for an overview. A FVE method thus has the same flexibility of a Finite Element Method (FEM) when it comes to solving on a complex domain with complex boundary conditions, while at the same time it preserves the local conservation properties like a FV method does.

Domain Decomposition Methods (DDM) form a class of effective parallel solvers for systems of algebraic equations arising from the FEM or the FDM discretization of PDEs. There exist a considerable amount of research work devoted towards the development of DDMs, that are available in the literature, cf. [2–4]

This work was partially supported by Polish Scientific Grant 2011/01/B/ST1/01179.

and the references therein. Additive Schwarz Methods (ASM) are among the most popular DDMs, however, a large number of their research are based on the FEM discretization, cf. e.g. [5–9] and the references therein, and only a few papers that consider the FV discretization, cf. [10, 11].

In this paper we propose a parallel ASM preconditioner for the nonsymmetric system of equations arising from the FV discretization of the self-adjoint elliptic second order problem in 2D. Preconditioned GMRES method is used to solve the resulting preconditioned system, cf. [12–14]. The preconditioner uses an abstract Schwarz framework where the solution space is decomposed into subspaces associated with the subdomains and the subdomain edges, and a coarse space associated with the skeleton which is the union of subdomain edges in the interior of the domain.

The remainder of this paper is organized as follows, in Sect. 2 we present the Finite Volume Element (FVE) discretization and describe its basic properties, and in Sect. 3 we recall the definition and basic convergence properties of the GMRES iterative method. Section 4 contains a description of the Additive Schwarz Method preconditioner, and in Sect. 5 we briefly discuss its implementation.

2 Finite Volume Element Discretization

Our aim is to find an approximation of the solution of the following self-adjoint second order elliptic differential problem:

$$\begin{aligned} -\nabla(A(x)\nabla u)(x) &= f(x) \qquad && x \in \Omega \\ u(s) &= 0 && s \in \partial\Omega \end{aligned}$$

where Ω is a polygonal domain on the plane, $f \in L^2(\Omega)$ and $A \in (W^{1,\infty}(\Omega))^4$ is a given symmetric matrix valued function such that

$$\exists\alpha > 0\, \forall x \in \Omega\, \forall\xi \in \mathbb{R}^2 \quad \xi^T A(x)\xi \geq \alpha\|\xi\|_2^2$$

i.e. we assume an uniform ellipticity of the problem.

The weak formulation is to find $u^* \in H_0^1(\Omega)$ such that

$$a(u^*, v) = f(v) \qquad \forall v \in H_0^1(\Omega), \tag{1}$$

where

$$\begin{aligned} a(u, v) &= \int_\Omega (\nabla u)^T A\nabla v\, dx, \\ f(v) &= \int_\Omega fv\, dx. \end{aligned} \tag{2}$$

We introduce a quasiuniform triangulation of the domain Ω: $T_h(\Omega) = T_h = \{\tau\}$ consisting of triangles τ, and let $h = \max_{\tau\in T_h} \operatorname{diam}(\tau)$ be the parameter of

T_h. Let Ω_h and $\partial\Omega_h$ be the sets of vertices of the triangles of T_h, which are in Ω and on $\partial\Omega$, respectively.

Let S_h be the finite element space of continuous functions which are piecewise linear over T_h and are equal to zero on $\partial\Omega$. The degrees of freedom are associated with the nodes which are the vertices of triangles, and we denote by Ω and $\partial\Omega_h$ respectively the set of nodes which are strictly inside Ω and $\partial\Omega$.

We introduce a dual mesh T_h^* of T_h in the following way, see also [15,16]. Let x_i, x_j and x_k be the vertices of a triangle τ. We connect x_τ (a point yet to be chosen inside τ) to the medians x_{ij} (the midpoints of the edges $\overline{x_i x_j}$ incident to x_i) by the straight lines $\gamma_{ij,\tau}$. For each interior vertex x_i, let ω_i be the polygon whose edges are $\gamma_{ij,\tau}$, for all τ sharing x_i as a vertex. Similarly, for each boundary vertex $x_i \in \partial\Omega$ the control volume ω_i is defined in the same way, but restricting it to Ω. See Fig. 1 for an illustration. The collection of all these control volumes constitute the dual mesh T_h^* of the domain Ω, i.e., $T_h^*(\Omega) = T_h^* = \{\omega_i\}_{x_i \in \Omega_h \cup \partial\Omega_h}$.

The interior point of an element τ may be chosen in different ways, for the present work, we choose the one which is most commonly used, namely the centroid, resulting in the so-called Donald mesh, cf. e.g. [15,16].

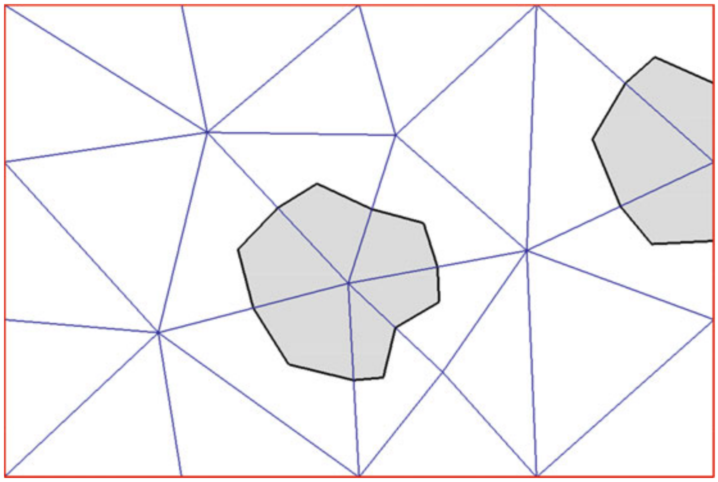


Fig. 1. Example of an interior- and boundary node control volume (shaded).

We now introduce the space S_h^* associated with the dual mesh T_h^*: let $S_h^* \subset L^2(\Omega)$ be the space of piecewise constant functions over T_h^*, taking zero values at the nodal points on $\partial\Omega$.

We introduce two nodal bases, one for S_h and one for S_h^*, the nodal basis $\{\phi_i\}_{x_i \in \Omega_h}$ of S_h, where $\phi_i \in S_h$ equals one at x_i and zero at the remaining nodes of Ω_h, and the basis $\{\psi_i\}_{x_i \in \Omega_h}$ of S_h^*, where $\psi_i \in S_h^*$ is one over the control volume $\omega_i \in T_h^*$ and zero over the remaining control volumes.

The two standard interpolation operators, $I_h : C(\Omega) \to S_h$ and $I_h^* : C(\Omega) \to S_h^*$, are then defined as

$$I_h u = \sum_{x_i \in \Omega_h} u(x_i)\phi_i \in S_h,$$
$$I_h^* u = \sum_{x_i \in \Omega_h} u(x_i)\psi_i \in S_h^*,$$

respectively. Let $a_{FV}(\cdot,\cdot)$ be the FV bilinear form defined as $a_{FV} : S_h \times S_h^* \to \mathbb{R}$ such that

$$a_{FV}(u,v) = -\sum_{x_i \in \Omega_h} v_i \int_{\partial V_i} A\nabla u \mathbf{n} ds \qquad u \in S_h, v \in S_h^*$$

and the nonsymmetric FV bilinear form $a_h(\cdot,\cdot)$ be defined as $a_h : S_h \times S_h \to \mathbb{R}$ such that

$$a_h(u,v) = a_{FV}(u, I_h^* v) \qquad u, v \in S_h.$$

The discrete FVE problem will then be formulated as follows: find $u_h \in S_h$ such that

$$a_h(u_h, v) = f(I_h^* v) \qquad \forall v \in S_h. \tag{3}$$

The following error estimate applies, cf. e.g. Sect. 3 in [16]:

$$\|u^* - u_h\|_{H^1(\Omega)} \leq C\, h \left(|u^*|_{H^2(\Omega)} + \|f\|_{L^2(\Omega)}\right)$$

provided that the solution $u^* \in H^2(\Omega)$. There also exist error estimates in the L^2 norm, cf. e.g. [16] and references therein.

By formulating the discrete problem in the standard nodal basis $\{\phi_i\}_{x_i \in \Omega_h}$, we get the following system of algebraic equations

$$B_h \boldsymbol{u} = \boldsymbol{f} \tag{4}$$

where $B_h = (a_h(\phi_i, \phi_j))_{i,j}$, $\boldsymbol{f} = (f_j)_{x_j \in \Omega_h}$ with $f_j = \int_\Omega f(x)\psi_i\, dx$ and $\boldsymbol{u} = (u_i)_i$ with $u_i = u_h(x_i)$. We have $u_h = \sum_{x_i \in \Omega_h} u_i \phi_i$. The resulting system is nonsymmetric which is in general very ill conditioned, and any standard iterative method may perform badly due to the ill-conditioning of the system and of the eigenspectrum.

The aim of this paper is therefore to solve such systems with the GMRES method (cf. Sect. 3, [13]) preconditioned by an Additive Schwarz Method (ASM) preconditioner (cf. [4]) in order to improve the convergence and the overall performance.

3 GMRES Method

In this section, we briefly recall the GMRES method and state some of its properties. Given a starting vector $u_0 \in S_h$, the aim is to solve iteratively the following system of linear equations,

$$Tu = g,$$

where T is an operator which is nonsingular but nonsymmetric, $g \in S_h$ is a given right-hand side vector, and $u \in S_h$ the solution vector. In our case the matrix formulation of T will be equal to $M_{ASM}^{-1}B_h$, where M_{ASM}^{-1} is the ASM preconditioner, cf. Sects. 4 and 5, while B_h is as in (4).

The GMRES method is a Krylov subspace iterative method. Let K_j be the j-th Krylov subspace with respect to the operator T and the residual vector $r_0 = g - Tu_0$, where u_0 is some vector, which is defined as

$$K_j = \mathrm{Span}(r_0, Tr_0, \ldots, T^{j-1}r_0) \qquad j = 1, 2, \ldots .$$

Starting with vector $u_0 \in S_h$, in the GMRES iteration, the j-th approximation can be defined as the solution of the following minimization problem: find $u_j \in u_0 + K_j$ such that

$$\|g - Tu_j\|_a = \min_{u \in u_0 + K_j} \|g - Tu\|_a \tag{5}$$

with the norm induced by the inner product $a(u, v)$, i.e.,

$$\|v\|_a := \sqrt{a(v, v)}.$$

The implementation of GMRES method with respect to the inner product $a(u, v)$ is done with the help of the Arnoldi process, cf. e.g. [17]

The rate of convergence of the GMRES method can be characterized in terms of the following two parameters,

$$\alpha_{min} = \min_{u \in S_h \setminus \{0\}} \frac{a(Tu, u)}{\|u\|_a^2},$$
$$\alpha_{max} = \max_{u \in S_h \setminus \{0\}} \frac{\|Tu\|_a}{\|u\|_a},$$

and is formulated in the following theorem (cf. [12]).

Theorem 1 (Eisenstat-Elman-Schultz). *If $\alpha_{min} > 0$, then the GMRES method is convergent and we have the following estimate:*

$$\|g - Tu_j\|_a \leq \left(1 - \frac{\alpha_{min}^2}{\alpha_{max}^2}\right)^{j/2} \|g - Tu_0\|_a.$$

where u_j is the j-th approximation of the GMRES method as defined in (5).

The original results which can be found in [12], were given for the GMRES in the standard l_2 inner product in $\mathbb{R}^N$, the lines of its proof however carry quite straightforwardly over to the case of $\|\cdot\|_a$ norm, cf. [2].

4 Additive Schwarz Method (ASM) Preconditioner

Our preconditioner is symmetric with respect to the bilinear form $a(\cdot,\cdot)$. Let Ω be partitioned into a collection of polygonal substructures Ω_k, such that

$$\overline{\Omega} = \bigcup_{k=1}^{N} \overline{\Omega}_k,$$

$$\overline{\Omega}_k \cap \overline{\Omega}_l = \begin{cases} \emptyset, \\ \text{a closed common edge,} \\ \text{a common vertex,} \end{cases}$$

which together form a coarse triangulation of Ω, which is shape regular in the sense of [18].

We also assume that the triangulation T_h is aligned with the subdomains Ω_k, i.e. any $\tau \in T_h$ is contained in only one subdomain, hence, each subdomain Ω_k inherits the local triangulation $T_h(\Omega_k) = \{\tau \in T_h : \tau \subset \Omega_k\}$. Let Γ_{kl} be the common edge between two substructures Ω_k and Ω_l, and $\overline{\Gamma} = (\bigcup_{k=1}^{N} \partial\Omega_k) \setminus \partial\Omega$ be the skeleton. Γ plays an important role in the construction of our method.

We define $\Omega_{k,h}$, $\Gamma_{kl,h}$ as the sets of vertices of triangles of T_h which are in Ω_k, Γ_{kl}, respectively.

We introduce our local subspaces $S_{h,k}$ as restrictions to $\overline{\Omega}_k$, of functions from S_h, i.e.,

$$S_{h,k} = \{u_{|\overline{\Omega}_k} : u \in S_h\} = \{v \in C(\overline{\Omega}_k) : v_{|\tau} \in P_1(\tau), \tau \in T_h(\Omega_k), v_{|\partial\Omega_k \cap \partial\Omega} = 0\},$$

and we let

$$S_{h,k}^0 = S_{h,k} \cap H_0^1(\Omega_k).$$

Here $P_1(\tau)$ is the space of linear polynomials defined over τ.

Let the local orthogonal projection operator $P_k : S_{h,k} \to S_{h,k}^0$, with respect to $a(u,v)$, be defined as

$$a_k(P_k u, v) = a_k(u, v) \qquad \forall v \in S_{h,k}^0,$$

and let the local discrete harmonic extension operator $H_k : S_{h,k} \to S_{h,k}$ be defined as

$$H_k u = u - P_k u.$$

Note that $H_k u$ satisfies the following variational problem:

$$\begin{aligned} a_k(H_k u, v) &= 0 \qquad \forall v \in S_{h,k}^0 \\ H_k u &= u \quad \text{on} \quad \partial\Omega_k. \end{aligned}$$

If $u_{|\overline{\Omega}_k} = H_k u \in S_{h,k}$ then we say that u is discrete harmonic in Ω_k. If for $u \in S_h$ we have $u_k := u_{|\overline{\Omega}_k}$ is discrete harmonic for all substructures then we call this function piecewise discrete harmonic. It is important to note that a discrete harmonic function in $S_{k,h}$ is uniquely defined by the values at the nodal points in $\partial\Omega_{k,h}$.

In the abstract framework of ASM we are required to introduce a decomposition of the global space S_h into the sum of smaller subspaces of S_h, as well as define local bilinear forms over these subspaces, cf. [3,4]. For the present work we consider only symmetric bilinear forms for the subspaces. The subspaces are defined below.

The coarse space V_0 is defined as the subspace of S_h of functions that are piecewise discrete harmonic and are linear over all edges $\Gamma_{kl} \subset \Gamma$. Note that the dimension of V_0 equals the number of the crosspoints (substructure vertices) which are not on the boundary of Ω.

A local edge based subspace V_{kl} associated with Γ_{kl}, which is the edge shared by the neighboring subdomains Ω_k and Ω_l, is defined as the space of piecewise discrete harmonic functions defined by the values of the functions at the nodes in $\Gamma_{kl,h}$ and zero values at the nodes in all remaining edges $\Gamma_{ij} \subset \Gamma$ which are not Γ_{kl}. Note that $u \in V_{kl}$ may have nonzero values only at nodes that are in $\Omega_{k,h} \cup \Gamma_{kl,h} \cup \Omega_{l,h}$.

The last collection of subspaces are the local subspaces V_k, $k = 1, \ldots, N$, with V_k being formed by the functions of $S^0_{k,h}$, extended by zero onto other subdomains, i.e.

$$V_k = \{u \in S_h : u_{|\overline{\Omega}_k} \in S^0_{k,h} \quad \text{and} \quad u_{|\Omega \setminus \Omega_k} = 0\}$$

We thus have that

$$S_h = V_0 + \sum_{k=1}^{N} V_k + \sum_{\Gamma_{kl} \subset \Gamma} V_{kl}.$$

For the local solves we choose $a(u, v)$ as the bilinear form, i.e. the symmetric bilinear form, see (2).

We define the projection like operator $T_0 : S_h \to V_0$, such that,

$$a(T_0 u, v) = a_h(u, v) \qquad \forall v \in V_0,$$

the local subspace operators, $T_k : S_h \to V_k$, such that,

$$a(T_k u, v) = a_h(u, v) \qquad \forall v \in V_k,$$

for $k = 1, \ldots, N$, and edge subspace operators $T_{kl} : S_h \to V_{kl}$

$$a(T_{kl} u, v) = a_h(u, v) \qquad \forall v \in V_{kl},$$

for all $\Gamma_{kl} \subset \Gamma$.

Finally, the additive Schwarz operator $T : S_h \to S_h$ is defined as follows,

$$T = T_0 + \sum_{k=1}^{N} T_k + \sum_{\Gamma_{kl}} T_{kl}.$$

Using the above settings, the problem (3) can be reformulated as the following equivalent system,

$$T u_h = g, \tag{6}$$

where

$$g = g_0 + \sum_{k=1}^{N} g_k + \sum_{\Gamma_{kl}} g_{kl}$$

with $g_0 = T_0 u_h$, $g_k = T_k u_h$ $k = 1, \ldots, N$, and $g_{kl} = T_{kl} u_h$ for all $\Gamma_{kl} \subset \Gamma$. In the following section we show that the preconditioned system has a better conditioning than the original system.

Remark 1. Note that g, the right-hand side of (6), can be computed without the knowledge of the solution vector u_h as shown in Sect. 5.

4.1 The Convergence of the Preconditioned GMRES Method

We present the main result of this paper, namely an estimate of the convergence rate of the GMRES method applied to our preconditioned system (3).

We have the following theorem.

Theorem 2. *There exists $h_0 > 0$ such that for all $h < h_0$*

$$\|Tu\|_a \le C\|u\|_a,$$
$$a(Tu, u) \ge c\left(1 + \log\left(\frac{H}{h}\right)\right)^{-2} a(u, u) \qquad \forall u \in S_h,$$

where C, c are positive constants independent of h and $H = \max_{k=1,\ldots,N} H_k$ where H_k is the diameter of Ω_k.

This theorem and Theorem 1 yield the following corollary.

Corollary 1. *There exists $h_0 > 0$ such that for all $h < h_0$ the GMRES method (5) applied to the system (6) is convergent and we have the following estimate:*

$$\|g - Tu_j\|_a \le \beta^{j/2}\|g - Tu_0\|_a.$$

where $0 \le \beta \le \left(1 - C\left(1 + \log\left(\frac{H}{h}\right)\right)^{-4}\right) < 1$ for a constant C which is independent of h and H and u_j is the j-th iteration of GMRES method.

Thus we see that GMRES method is convergent with the rate only weakly polylogarithmically dependent on H/h.

As mentioned earlier, we can rewrite the problem (6) in the matrix formulation as

$$M_{ASM}^{-1} B_h \boldsymbol{u} = M_{ASM}^{-1} \boldsymbol{f}$$

with M_{ASM}^{-1} being our ASM parallel preconditioner, see (4).

Remark 2. While the matrix B_h is nonsymmetric, the ASM preconditioner itself in this paper is symmetric since we are using the symmetric bilinear form $a(\cdot, \cdot)$ (cf. (2)) for all of our subspace problems. Alternatively, we could construct an ASM preconditioner in the same way, but taking the nonsymmetric bilinear form $a_h(\cdot, \cdot)$ for the subspace problems. The resulting ASM preconditioner will then be nonsymmetric, however, it will have the similar estimate.

5 Implementation

In this section, we briefly discuss the implementation of our ASM preconditioner. We propose to use the GMRES method to solve the preconditioned system (6), however, for the simplicity of presentation, we describe the ideas of implementation through Richardson's iteration.

Note that by Theorem 2 the operator T is positive definite with respect to $a(u,v)$ over S_h, i.e. $a(Tu,u) > 0$ for $u \neq 0$, thus the Richardson method is convergent for some positive values of its parameter τ.

The Richardson iterative method with the parameter τ is defined as follows: Take any $u^{(0)}$ and iterate until convergence:

$$\begin{aligned} u^{(n+1)} &= u^{(n)} - \tau\,(T(u^{(n)}) - g) \\ &= u^{(n)} - \tau\, T(u^{(n)} - u_h^*)\ , \qquad n \geq 0. \\ &= u^{(n)} - \tau\, r^{(n)} \end{aligned}$$

Computing of $r^{(n)} = T(u^{(n)}) - g$ requires solving the following problems:

▷ **Local subdomain problem**:
Compute $r_k \in V_k$ $k = 1, \ldots, N$ such that

$$\begin{aligned} a(r_k, v) &= a(T_k(u^{(n)} - u_h^*), v) \\ &= a_h(u^{(n)}, v) - f(v) \qquad \forall v \in V_k. \end{aligned}$$

▷ **Local edge problem**:
Compute $r_{kl} \in V_{kl}$ for all $\Gamma_{kl} \subset \Gamma$ such that

$$a(r_{kl}, v) = a(T_{kl}(u^{(n)} - u_h^*), v) = a_h(u^{(n)}, v) - f(v) \qquad \forall v \in V_{kl},$$

▷ **Coarse problem**:
Compute $r_0 \in V_0$ such that

$$a(r_0, v) = a(T_0(u^{(n)} - u_h^*), v) = a_h(u^{(n)}, v) - f(v) \qquad \forall v \in V_0,$$

Then

$$r^{(n)} = r_0 + \sum_{\gamma_{kl} \subset \Gamma} r_{kl} + \sum_{x \in \mathcal{V}} \sum_{s=1,2} r_{x,s}.$$

Note that all these problems are independent so they can be solved in parallel.

The local subdomain problems are solved locally on their respective subdomains, the edge problems are solved locally over subdomain pairs, each pair sharing an edge. The coarse problem is global but its dimension is low as long the number of substructures is not too large.

References

1. Lin, Y., Liu, J., Yang, M.: Finite volume element methods: an overview on recent developments. Int. J. Num. Anal. Mod. **4**(1), 14–34 (2013)

2. Cai, X.C., Widlund, O.B.: Domain decomposition algorithms for indefinite elliptic problems. SIAM J. Sci. Statist. Comput. **13**(1), 243–258 (1992)
3. Smith, B.F., Bjørstad, P.E., Gropp, W.D.: Domain Decomposition: Parallel Multilevel Methods For Elliptic Partial Differential Equations. Cambridge University Press, Cambridge (1996)
4. Toselli, A., Widlund, O.: Domain Decomposition Methods-Algorithms and Theory. Springer Series in Computational Mathematics, vol. 34. Springer, Berlin (2005)
5. Dryja, M., Widlund, O.B.: Schwarz methods of Neumann-Neumann type for three-dimensional elliptic finite element problems. Comm. Pure Appl. Math. **48**(2), 121–155 (1995)
6. Brenner, S.C., Wang, K.: Two-level additive Schwarz preconditioners for C^0 interior penalty methods. Numer. Math. **102**(2), 231–255 (2005)
7. Mandel, J., Brezina, M.: Balancing domain decomposition for problems with large jumps in coefficients. Math. Comp. **65**(216), 1387–1401 (1996)
8. Griebel, M., Oswald, P.: On the abstract theory of additive and multiplicative Schwarz algorithms. Numer. Math. **70**(2), 163–180 (1995)
9. Marcinkowski, L.: A balancing Neumann-Neumann method for a mortar finite element discretization of a fourth order elliptic problem. J. Numer. Math. **18**(3), 219–234 (2010)
10. Chou, S.H., Huang, J.: A domain decomposition algorithm for general covolume methods for elliptic problems. J. Numer. Math. **11**(3), 179–194 (2003)
11. Zhang, S.: On domain decomposition algorithms for covolume methods for elliptic problems. Comput. Meth. Appl. Mech. Engrg. **196**(1–3), 24–32 (2006)
12. Eisenstat, S.C., Elman, H.C., Schultz, M.H.: Variational iterative methods for nonsymmetric systems of linear equations. SIAM J. Numer. Anal. **20**(2), 345–357 (1983)
13. Saad, Y., Schultz, M.H.: GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Statist. Comput. **7**(3), 856–869 (1986)
14. Xu, J., Cai, X.C.: A preconditioned GMRES method for nonsymmetric or indefinite problems. Math. Comp. **59**(200), 311–319 (1992)
15. Huang, J., Xi, S.: On the finite volume element method for general self-adjoint elliptic problems. SIAM J. Numer. Anal. **35**(5), 1762–1774 (1998)
16. Ewing, R.E., Lin, T., Lin, Y.: On the accuracy of the finite volume element method based on piecewise linear polynomials. SIAM J. Numer. Anal. **39**(6), 1865–1888 (2002)
17. Cai, X.C.: Some domain decomposition algorithms for nonselfadjoin elliptic and parabolic partial differential equations. Ph.D. thesis, Courant Institute, New York (1989)
18. Brenner, S.C.: The condition number of the Schur complement in domain decomposition. Numer. Math. **83**(2), 187–203 (1999)

Preconditioning Iterative Substructuring Methods Using Inexact Local Solvers

Piotr Krzyzanowski(✉)

Institute of Applied Mathematics, University of Warsaw, Banacha 2, 02-097 Warszawa, Poland
piotr.krzyzanowski@mimuw.edu.pl

Abstract. We consider several block preconditioners for iterative substructuring algorithms with inexact subdomain solvers, including incomplete Cholesky and V-cycle multigrid. Numerical results show that block triangular preconditioners are very competitive and in certain cases outperform presently used preconditioners based on full block triangular decomposition.

Keywords: Preconditioning · Iterative substructuring · Domain decomposition · Block preconditioner

1 Introduction

In recent years there has been significant progress made towards development of fast and reliable parallel solvers for very large algebraic linear systems which arise from discretizations of elliptic PDEs,

$$-\operatorname{div}(K(x)\nabla U(x)) + C(x)U(x) = F(x), \qquad x \in \Omega,$$

where $\Omega \subset R^d$ and K is a positive definite $d \times d$ symmetric matrix, and C is a nonnegative scalar function defined on Ω. Discretization of the above PDE with suitable boundary conditions by using Lagrangian finite elements leads to a linear, ill-conditioned system of equations

$$\mathcal{A}u = f, \tag{1}$$

with a sparse, symmetric and positive definite matrix $\mathcal{A}$ of very large dimension.

1.1 Substructuring Methods

Among the most successful parallel solution methods for (1) there are those based on the iterative substructuring paradigm. In this approach, one provides

This work has been supported by Polish National Science Centre research grant 2011/01/B/ST1/01179.

a decomposition of triangulation (and hence the domain Ω) into nonoverlapping subdomains Ω_i, $i = 1, \ldots, K$ ("substructures") such that each Ω_i is a union of triangulation elements and

$$\Omega = \cup_{i=1}^K \Omega_i.$$

Introducing the interface between subdomains,

$$\Gamma = \cup_{i=1}^K \partial\Omega_i \setminus \partial\Omega,$$

every unknown of the discretized PDE, which in case of Lagrangian elements corresponds to the value of the discrete solution at certain point of the triangulation element, belongs to exactly one of $K+1$ sets: either to the interior of one of Ω_i's or to the interface Γ. We can therefore partition the vector u of the unknowns into smaller vectors, which we shall denote $u_1^I, \ldots, u_K^I$ and u^Γ, with u_i^I corresponding to unknowns in the interior of Ω_i and u^Γ associated with unknowns on Γ. The discrete system (1) can then be written in a block form

$$\begin{pmatrix} A_1^{II} & & & {A_1^{\Gamma I}}^T \\ & \ddots & & \vdots \\ & & A_K^{II} & {A_K^{\Gamma I}}^T \\ A_1^{\Gamma I} & & A_K^{\Gamma I} & A^{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} u_1^I \\ \vdots \\ u_K^I \\ u^\Gamma \end{pmatrix} = \begin{pmatrix} f_1^I \\ \vdots \\ f_K^I \\ f^\Gamma \end{pmatrix}, \tag{2}$$

the structure of which promotes parallelization. Indeed, one first eliminates in parallel all interior unknowns performing a block Gaussian elimination on $u_1^I, \ldots, u_K^I$ and then solves the Schur complement system

$$Su^\Gamma = g, \tag{3}$$

where

$$S = A^{\Gamma\Gamma} - \sum_{i=1}^K A_i^{\Gamma I}(A_i^{II})^{-1}{A_i^{\Gamma I}}^T$$

and $g = f^\Gamma - \sum_{i=1}^K A_i^{\Gamma I}(A_i^{II})^{-1} f_i^I$. After u^Γ is found, one then in parallel backsolves for all $u_1^I, \ldots, u_K^I$.

When the size of u^Γ is still very large, it may be infeasible to construct matrix S explicitly, which in addition will be denser than the original system matrix A. In iterative substructuring thus, one turns to iterative solution of (3), usually using some Krylov method. Then it is not necessary to form S directly, because Krylov methods require only a procedure which computes the product Sx on a supplied vector x. This product can be obtained again in a highly parallel fashion, since

$$Sx = A^{\Gamma\Gamma}x - \sum_{i=1}^K A_i^{\Gamma I} y_i \tag{4}$$

where $y_i = (A_i^{II})^{-1}{A_i^{\Gamma I}}^T x$ can be computed independently.

1.2 Preconditioning in Iterative Substructuring

While the Schur complement matrix S is symmetric positive definite and is better conditioned than A, its condition number still grows with the number of the unknowns and therefore it is necessary to precondition S with some spectrally equivalent operator S_0. There has been a lot of research done towards this direction and we can refer the reader to monographs [1–3] and the literature therein. Many of these preconditioners are highly parallel and utilize the same original decomposition of the domain into substructures. Among them there are wirebasket methods [4] and hierarchical basis methods [5] which require quite detailed information about the underlying geometry but have almost optimal convergence properties.

There is still some place for performance improvement on the side of parallel solution of the local systems with matrices A_i^{II}, $i = 1, \ldots, K$. Indeed, in the procedure described in Sect. 1.1 it has implicitly been assumed that these local systems are always solved exactly, either by a direct sparse solver, or by some iterative method which reduces the error down to the machine precision. When the size of local problems is large, this can become very expensive, especially for problems in 3D, for which direct sparse solvers suffer from high amount of fill-in in Cholesky factors of A_i^{II}'s. In iterative substructuring, even small efficiency improvement in the local problem solution can have an impact on the overall solver performance, because local problems have to be solved every time the product Sx is needed, that is, on every iteration of the method.

1.3 Iterative Substructuring with Inexact Local Solvers

This numerical study thus focuses on using *inexact* local solvers in order to reduce the cost of solving the local systems and finally obtain a more effective solution method of (1). We shall replace local solves $(A_i^{II})^{-1}u^I$ by certain less expensive $(A_{i0}^{II})^{-1}u^I$, where $(A_{i0}^{II})^{-1}$ are symmetric positive definite operators (the very A_{i0}^{II} matrices will never explicitly be formed). In this case we can no longer use the $(K+1)$ step block Gaussian procedure described in Sect. 1.1 and all unknowns will have to enter the iterative process [6]. In order to simplify the notation we will gather all interior unknowns $(u_1^I, \ldots, u_K^I)$ into one vector u^I, obtaining a 2×2 block system

$$\mathcal{A}\begin{pmatrix} u^I \\ u^\Gamma \end{pmatrix} \equiv \begin{pmatrix} A^{II} & {A^{\Gamma I}}^T \\ A^{\Gamma I} & A^{\Gamma\Gamma} \end{pmatrix}\begin{pmatrix} u^I \\ u^\Gamma \end{pmatrix} = \begin{pmatrix} f^I \\ f^\Gamma \end{pmatrix} \tag{5}$$

and denote

$$A_0^{II} = \operatorname{diag}(A_{0,1}^{II}, \ldots, A_{0,K}^{II}) = \begin{pmatrix} A_{0,1}^{II} & & \\ & \ddots & \\ & & A_{0,K}^{II} \end{pmatrix}.$$

We shall keep in mind that both the multiplication $A^{II}u^I$ and the solution a system with A_0^{II} can be done fully in parallel due to the block diagonal structure of both A^{II} and A_0^{II}.

2 Preconditioning with Inexact Local Solvers

In what follows we shall concentrate on block preconditioners $\mathcal{P}$, and the preconditioned system will be $\mathcal{P}^{-1}\mathcal{A}$. Deriving from the block LDL^T decomposition,

$$\mathcal{A} = \begin{pmatrix} A^{II} & A^{\Gamma I^T} \\ A^{\Gamma I} & A^{\Gamma\Gamma} \end{pmatrix} = \begin{pmatrix} I & \\ A^{\Gamma I} A^{II^{-1}} & I \end{pmatrix} \begin{pmatrix} A^{II} & \\ & S \end{pmatrix} \begin{pmatrix} I & A^{II^{-1}} A^{\Gamma I^T} \\ & I \end{pmatrix}, \quad (6)$$

most of the literature on this subject [1,2,6–8], considers the following preconditioner based on full block decomposition:

$$\mathcal{P}_G = \begin{pmatrix} I & \\ A^{\Gamma I} A_0^{II^{-1}} & I \end{pmatrix} \begin{pmatrix} A_0^{II} & \\ & S_0 \end{pmatrix} \begin{pmatrix} I & A_0^{II^{-1}} A^{\Gamma I^T} \\ & I \end{pmatrix}. \quad (7)$$

Here, we assume that systems with A_0^{II} and S_0 are cheaper to solve than exact blocks A^{II} and S. Because

$$\mathcal{P}_G^{-1} = \begin{pmatrix} I & -A_0^{II^{-1}} A^{\Gamma I^T} \\ & I \end{pmatrix} \begin{pmatrix} A_0^{II^{-1}} & \\ & S_0^{-1} \end{pmatrix} \begin{pmatrix} I & \\ -A^{\Gamma I} A_0^{II^{-1}} & I \end{pmatrix}, \quad (8)$$

application of $\mathcal{P}_G^{-1}$ to a vector can be arranged in such a way that one solve with S_0 and two solves with A_0^{II} are required [3]. In order to make $\mathcal{P}_G$ a good preconditioner for $\mathcal{A}$, one can choose A_0^{II} as a block diagonal matrix made up from approximate solvers for $A_{0,k}^{II}$ which can easily be applied in parallel. This method has been analyzed and backed up by convincing numerical experiments, see e.g. [3,6,9].

Motivated by recent advances in a similar field — block preconditioning of saddle point problems [10]— we will here consider three (five, counting sign variations) more candidates for a preconditioner, based on a part of decomposition (7):

$$\mathcal{P}_L^{\pm} = \begin{pmatrix} A_0^{II} & \\ \pm A^{\Gamma I} & S_0 \end{pmatrix}, \qquad \mathcal{P}_D = \begin{pmatrix} A_0^{II} & \\ & S_0 \end{pmatrix}, \qquad \mathcal{P}_U^{\pm} = \begin{pmatrix} A_0^{II} & \pm A^{\Gamma I^T} \\ & S_0 \end{pmatrix}. \quad (9)$$

Observe that solving a system with any of $\mathcal{P}_L, \mathcal{P}_D, \mathcal{P}_U$ takes only one solve with both S_0 and A_0^{II} and thus the overall cost of local solves part in this case is two times smaller than in the case of $\mathcal{P}_G$. Assuming that the cost of each iteration is dominated by the application of the preconditioner, one can conclude that the cost of one iteration with $\mathcal{P}_L, \mathcal{P}_D, \mathcal{P}_U$ is roughly between 1/2 and 2/3 of the cost when $\mathcal{P}_G$ is employed. This rough estimate should in practice be balanced with parallelization and communication considerations and with the actual complexity of application of S_0^{-1}.

According to the author's knowledge, it is an open question if $\mathcal{P}_L, \mathcal{P}_D, \mathcal{P}_U$ are a viable alternative to $\mathcal{P}_G$ in the context of iterative substructuring. Our aim here is to perform a numerical comparison of these preconditioners on a model problem to obtain at least a partial answer to this question.

2.1 Krylov Iterative Method Considerations

By definition, $\mathcal{P}_G$ and $\mathcal{P}_D$ are symmetric and positive definite, so it is natural to use them as preconditioners in the preconditioned conjugate gradient method (PCG). On the other hand, $\mathcal{P}_L$ and $\mathcal{P}_U$ are nonsymmetric so it seems one has to use more memory–consuming method such as the GMRES. However, it has recently been observed [11–13] that it is possible to symmetrize the preconditioned system with a means of an additional block diagonal matrix, $\mathcal{H}_L$ for $\mathcal{P}_L$ and $\mathcal{H}_U$ for $\mathcal{P}_U$, where

$$\mathcal{H}_L^{\pm} = \begin{pmatrix} A_0^{II} \mp A^{II} & \\ & S_0 \end{pmatrix}, \qquad \mathcal{H}_U^{\pm} = \begin{pmatrix} A_0^{II} & \\ & S_0 \mp A^{\Gamma\Gamma} \end{pmatrix}. \tag{10}$$

So, if only $\mathcal{H}_L$ or $\mathcal{H}_U$ is positive definite (which, in case of matrix subtraction in (10), will in general require scaling either A_0^{II} or S_0 by some constant), one can apply a three–term–recurrence–based Krylov space method, such as the conjugate residual method [14], to then symmetric matrix $\mathcal{H}_L\mathcal{P}_L^{-1}\mathcal{A}$ or $\mathcal{H}_U\mathcal{P}_U^{-1}\mathcal{A}$. It should be mentioned here that the algorithm can be arranged in such a way that only one solve with both A_0^{II} and S_0 is required per iteration and the matrices A_0^{II} and S_0 never need to be formed or applied to a vector, see [13] or [10] for details.

Let us finally remark that in the case of "best" preconditioners $A_0^{II} = A^{II}$ and $S_0 = S$, we have that $\mathcal{P}_G^{-1}\mathcal{A}$ and $(\mathcal{P}_U^{+})^{-1}\mathcal{A}$ converge in one or two iterations, respectively.

3 Numerical Experiments

We will consider a simple model problem: the Poisson equation,

$$-\Delta u = f$$

in a rectangle Ω, supplemented with zero Dirichlet boundary conditions. The rectangle is composed of $N_x \times N_y$ unit squares, which will play the role of subdomains Ω_i, so that the total number of subdomains is $K = N_x \cdot N_y$. Each Ω_i is then triangulated with a uniform mesh (consisting of squares of size h halved into two triangles); the resulting triangulation of Ω is conforming and uniform, too. The Laplacian is finally discretized using piecewise linear finite element functions associated with the triangulation, resulting in a system of linear algebraic equations of total N unknowns (N depending linearly on N_x and N_y).

For the discretized right hand side f we will always supply the same vector with random values uniformly distributed in $(0, 1)$. The initial approximation $\tilde{u}$ will always be set to zero. The iteration will be stopped whenever the initial residual, $||f - \mathcal{A}\tilde{u}||_2$, is reduced 10^6 times.

Let us begin with a series of convergence speed tests for various choices of the preconditioners and their diagonal blocks.

As for the local solvers $(A^{II}_{0,i})^{-1}$ we will experiment with:

1. exact solve with A^{II}_i;
2. solvers based on incomplete Cholesky factorization of A^{II}_i, with drop parameter δ; the resulting A^{II}_0 will be denoted IC(δ);
3. solvers based on p multigrid V–cycle iterations for A^{II}_i with 3 damped Jacobi pre- and postsmoothings [15]; the resulting A^{II}_0 will be denoted MGV$(3,p)$.

For the approximate Schur complement system solvers S_0^{-1} we will consider:

1. exact solve with $S_0 = \alpha S$, where $S = A^{\Gamma\Gamma} - A^{\Gamma I}(A^{II})^{-1}A^{\Gamma I^T}$ is computed directly and $\alpha > 0$.
2. for two subdomains case, the so-called J preconditioner, [1,2,16], based on the square root of S.

We do not employ any further scaling of the above preconditioning blocks.

We will report both iteration counts (m) and solution times (t, in seconds) obtained for a sequential implementation of the above algorithms in MATLAB 7.1 on a personal computer with an Intel Core 2 Duo 2.4 GHz processor. We report solver's performance for best suited iterative method, that is, the preconditioned conjugate gradient method (PCG) for $\mathcal{P}_D$ and $\mathcal{P}_G$ and the preconditioned conjugate residual method (PCR) for $\mathcal{P}_U^+$ (the performance of $\mathcal{P}_U^-$ turns out inferior to $\mathcal{P}_U^+$; the performance of $\mathcal{P}_L^{\pm}$ is similar to that of $\mathcal{P}_U^{\pm}$; we do not report results for either case here).

For a problem on a 320×320 grid decomposed into 10×10 subdomains, we obtained results summarized in Table 1.

Experimental results for two subdomains and 512×256 grid are reported in Table 2.

It turns out that for the performance of block preconditioners with inexact solvers it is quite important that the local solvers are accurate enough: compare $IC(L, 10^{-2})$, for which the process diverges, and $IC(L, 10^{-5})$, which produces results very similar to using an exact solver. Similarly, the choice of the S_0 block also influences the convergence. In all experiments the convergence of $\mathcal{P}_D$ was dramatically slower than the competition, even in the case when exact solves were applied inside $\mathcal{P}_D$. As for the comparison between $\mathcal{P}_U^+$ and $\mathcal{P}_G$, it turns out that when the cost of applying S_0^{-1} was relatively cheap, then $\mathcal{P}_U^+$ performed better in terms of the overall cost, cf. Table 2: while the number of iterations was larger than for $\mathcal{P}_G$, the savings on the cost of every iteration were substantial enough to give the triangular preconditioner an edge over $\mathcal{P}_G$.

Preconditioned iterative substructuring algorithms allow for several levels of parallelism. On the base level, there is the domain decomposition paradigm (2), which allows one to solve all local systems $A^{II}_{0,1}, \dots A^{II}_{0,K}$ in (9) independently, where K is the number of subdomains. On the lower level, each local problem can, if only sufficiently large, again be solved using some parallel method.

In order to assess the parallel performance, we ran several experiments using PETSc 3.4.0 [17] toolkit which implements block preconditioners via the FieldSplit flexible class of preconditioners. Developed for multiphysics applications [18],

Table 1. Number of iterations m and computing time t for a problem on a 320×320 grid decomposed into 10×10 subdomains.

	m	t	A_0^{II}	S_0
$\mathcal{P}_G$	1	0.35	A_{II}	S
$\mathcal{P}_U^+$	2	1.33	A_{II}	S
$\mathcal{P}_D$	784	249.58	A_{II}	S
$\mathcal{P}_G$	3	4.40	$MGV(3,4)$	S
$\mathcal{P}_U^+$	8	9.38	$MGV(3,4)$	S
$\mathcal{P}_D$	793	751.21	$MGV(3,4)$	S
$\mathcal{P}_G$	3	1.24	$IC(10^{-5})$	S
$\mathcal{P}_U^+$	7	3.36	$IC(10^{-5})$	S
$\mathcal{P}_D$	799	280.87	$IC(10^{-5})$	S
$\mathcal{P}_G$	31	7.05	$IC(10^{-3})$	S
$\mathcal{P}_U^+$	92	25.43	$IC(10^{-3})$	S
$\mathcal{P}_G$	231	51.18	$IC(10^{-2})$	S
$\mathcal{P}_U^+$	280	76.53	$IC(10^{-2})$	S
$\mathcal{P}_G$	2	0.74	A_{II}	$4S$
$\mathcal{P}_U^+$	2	1.41	A_{II}	$4S$
$\mathcal{P}_G$	4	6.08	$MGV(3,4)$	$4S$
$\mathcal{P}_U^+$	7	8.72	$MGV(3,4)$	$4S$
$\mathcal{P}_G$	4	1.50	$IC(10^{-5})$	$4S$
$\mathcal{P}_U^+$	7	3.47	$IC(10^{-5})$	$4S$

FieldSplit offers a framework in which it is possible to use preconditioners based on either block relaxation, or block factorization — the latter using Schur complement S or its approximation. Within blocks, either direct or inexact iterative solvers can be used.

For three main types preconditioners: full-decomposition-based block $\mathcal{P}_G$, triangular block $\mathcal{P}_U^+$ and diagonal block preconditioner $\mathcal{P}_D$ as well, we measured time τ (in seconds, averaged over the course of the iteration) required to perform one iteration of the iterative solver, on an SMP machine equipped with 64 GB of RAM and 24, six–core, Intel Xeon 2.4 GHz processors using PETSc library. On such a machine one cannot expect massive scalability, so in our experiments we limited ourselves only to moderate number of MPI processes in our PETSc code, equal to the number of subdomains, $K = 16$.

For the local inexact solvers A_0^{II} we used

- Cholesky factorization with MUMPS parallel solver (referred to as A^{II} in the following tables) — the most expensive "preconditioner", which solves local problems exactly;
- 10 iterations of the CG method for A^{II}, preconditioned with an additive Schwarz method with PETSc default settings (denoted ASM(II)) — a relatively inexpensive, low accuracy preconditioner.

As the interface preconditioner, S_0, we tested

- simple point Jacobi solver for (partially assembled) Schur complement matrix — a very cheap and highly inaccurate preconditioner

Table 2. Number of iterations m and computing time t for a problem on a 512×256 grid decomposed into two subdomains.

	m	t	A_0^{II}	S_0
$\mathcal{P}_G$	4	5.36	A_{II}	J
$\mathcal{P}_U^+$	5	5.24	A_{II}	J
$\mathcal{P}_D$	224	159.82	A_{II}	J
$\mathcal{P}_G$	15	9.81	$MGV(3,4)$	J
$\mathcal{P}_U^+$	21	8.33	$MGV(3,4)$	J
$\mathcal{P}_D$	242	81.37	$MGV(3,4)$	J
$\mathcal{P}_G$	10	6.97	$IC(10^{-5})$	J
$\mathcal{P}_U^+$	12	5.76	$IC(10^{-5})$	J
$\mathcal{P}_D$	235	86.04	$IC(10^{-5})$	J

Table 3. Comparison of the average wall-clock time required to perform one iteration of the solver for a problem of $2.6 \cdot 10^5$ unknowns decomposed into 16 square subdomains mapped onto 16 processors.

	ASM(II) + ASM($\Gamma\Gamma$)	A^{II} + Jacobi	ASM(II) + Jacobi
$\mathcal{P}_D$	7.4	0.8	0.16
$\mathcal{P}_U^+$	6.9	0.8	0.17
$\mathcal{P}_G$	7.5	1.4	0.31

– 100 iterations of CG method for S preconditioned with an additive Schwarz method with PETSc default settings (denoted ASM($\Gamma\Gamma$)) — due to some limitations of FieldSplit, we used this setting to simulate the use of an expensive, yet modereately accurate preconditioner to S.

Table 3 summarizes the results for a problem with total $2.6 \cdot 10^5$ unknowns decomposed into 16 square subdomains. The combination A^{II} + ASM($\Gamma\Gamma$) turned out too costly to run on the machine under consideration.

Clearly, the cost of applying the preconditioner depends on its form and the cost of applying $(A_0^{II})^{-1}$ and S_0^{-1}. When the cost of local solves is dominating, then triangular solver $\mathcal{P}_U^+$ is almost twice less expensive than the full one, $\mathcal{P}_G$ and essentially as expensive as the diagonal one. It turns out then, that the additional cost of multiplication by the off–diagonal block is negligible in parallel implementation and in this sense our conclusions from sequential runs provided in the first table also extend to parallel implementations. This confirms our rough cost estimates from Sect. 2. On the other hand, when the cost of S_0^{-1} is prohibitively large (the case "ASM + ASM"), then the cost of applying the preconditioner turns more or less the same for all types of preconditioners (surprisingly poor performance of $\mathcal{P}_D$ in this case can be, most probably, be attributed to uneven load on the machine).

In practice, one will rather use more efficient and less expensive S_0, such as the balancing Neumann–Neumann method, which would then favor triangular preconditioners. Performance analysis of such a method is more complicated, however, and will be the subject of a forthcoming paper.

4 Conclusions

In this paper, we introduced block triangular preconditioners as a viable alternative to full block decomposition when using inexact local solvers in iterative substructuring of symmetric, elliptic PDEs. It turns out that triangular preconditioners are not only cheaper to apply, but despite their lower convergence speed, as compared to full block decomposition based their counterparts, in certain cases they are more efficient in terms of the wall-clock time. Although triangular preconditioning makes the problem apparently nonsymmetric, its symmetry can be restored through a custom inner product, which makes it possible to iterate using a short-term recurrence Krylov method, such as the PCR. It is also remarkable, how poorly diagonal preconditioners behave in this situation.

Acknowledgments. This research has been supported by Polish National Science Centre grant, number 2011/01/B/ST1/01179.

References

1. Toselli, A., Widlund, O.: Domain decomposition methods-algorithms and theory. Springer Series in Computational Mathematics, vol. 34. Springer, Berlin (2005)
2. Smith, B.F., Bjørstad, P.E., Gropp, W.D.: Domain Decomposition. Cambridge University Press, Cambridge (1996). (Parallel multilevel methods for elliptic partial differential equations.)
3. Mathew, T.P.A.: Domain decomposition methods for the numerical solution of partial differential equations. Lecture Notes in Computational Science and Engineering, vol. 61. Springer, Berlin (2008)
4. Dryja, M., Smith, B., Widlund, O.: Schwarz analysis of iterative substructuring algorithms for elliptic problems in three dimensions. SIAM J. Num. Anal. **31**(6), 1662–1694 (1994)
5. Bramble, J.H., Pasciak, J.E., Schatz, A.H.: The construction of preconditioners for elliptic problems by substructuring. IV. Math. Comp. **53**(187), 1–24 (1989)
6. Smith, B.F.: A parallel implementation of an iterative substructuring algorithm for problems in three dimensions. SIAM J. Sci. Comput. **14**(2), 406–423 (1993)
7. Bramble, J.H., Pasciak, J.E., Vassilev, A.T.: Analysis of non-overlapping domain decomposition algorithms with inexact solves. Math. Comp. **67**(221), 1–19 (1998)
8. Haase, G., Langer, U., Meyer, A.: The approximate Dirichlet domain decomposition method. I. An algebraic approach. Computing **47**(2), 137–151 (1991)
9. Haase, G., Langer, U., Meyer, A.: The approximate Dirichlet domain decomposition method. II. Applications to 2nd-order elliptic BVPs. Computing **47**(2), 153–167 (1991)
10. Krzyżanowski, P.: Block preconditioners for saddle point problems resulting from discretizations of partial differential equations. In: Axelsson, O., Karatson, J. (eds.) Efficient Preconditioned Solution Methods for Elliptic Partial Differential Equations. Bentham Publishers (2011)

11. Bramble, J.H., Pasciak, J.E.: A preconditioning technique for indefinite systems resulting from mixed approximations of elliptic problems. Math. Comp. **50**(181), 1–17 (1988)
12. Stoll, M., Wathen, A.: Combination preconditioning and the Bramble-Pasciak$^+$ preconditioner. SIAM J. Matrix Anal. Appl. **30**(2), 582–608 (2008)
13. Krzyżanowski, P.: On block preconditioners for saddle point problems with singular or indefinite (1, 1) block. Numer. Linear Algebra Appl. **18**(1), 123–140 (2011)
14. Hackbusch, W.: Elliptic differential equations. Springer Series in Computational Mathematics, vol. 18. Springer, Berlin (1992) (Theory and numerical treatment, Translated from the author's revision of the 1986 German original by R. Fadiman and P.D.F. Ion.)
15. Hackbusch, W.: Multigrid Methods and Applications. Springer, Berlin (1985)
16. Dryja, M.: A finite element-capacitance method for elliptic problems on regions partitioned into subregions. Numer. Math. **44**(2), 153–168 (1984)
17. Smith, B., Gropp, W., McInnes, L.: PETSc 2.0 users manual. Technical report ANL-95/11, Argonne National Laboratory. ftp://www.mcs.anl/pub/petsc/manual.ps (1997)
18. Brown, J., Knepley, M.G., May, D.A., McInnes, L.C., Smith, B.: Composable linear solvers for multiphysics. In: International Symposium on Parallel and Distributed Computing, pp. 55–62 (2012)

Additive Schwarz Method for Nonsymmetric Local Discontinuous Galerkin Discretization of Elliptic Problem

Filip Z. Klawe(✉)

Institute of Applied Mathematics and Mechanics,
University of Warsaw, Banacha 2, 02-097 Warszawa, Poland
fzklawe@mimuw.edu.pl

Abstract. In this paper we design two-level additive Schwarz method method for non-symmetric, elliptic problem in two dimensions with the use of discerization by local discontinuous Galerkin method (LDG). To construct the preconditioner, we use the domain decomposition method. We also want to show the result of numerical tests regarding to this preconditioner. Condition of the preconditioned system does not depend on the size of fine mesh h, but only on the ratio of the coarse mesh size H and the overlap measure δ.

Keywords: Parallel preconditioner · Local discontinuous Galerkin · Convection-diffusion

1 Introduction

The discontinuous Galerkin method (DG) was introduced by Reed and Hill [1] in 1973 for hyperbolic equations. Since that time many DG methods have been developed for various types of PDEs. The local discontinuous Galerkin method (LDG) was introduced by Cockburn and Shu in [2] and further studied in [3,4] for convection-diffusion problem.

Let us consider the problem

$$\begin{cases} -\Delta u + 2\boldsymbol{b}\cdot\nabla u + cu = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases} \tag{1}$$

where Ω is a bounded polygonal domain in $\mathbb{R}^2$ with boundary $\partial\Omega$. The function f belongs to $L^2(\Omega)$. Vector function $\boldsymbol{b}$ is constant and function c belongs to $L^\infty(\Omega)$. Moreover, we assume that the solution of problem (1) exist.

The author is a PhD student of the International PhD Projects Programme of Foundation for Polish Science operating within the Innovative Economy Operational Programme 2007–2013 funded by the European Regional Development Fund (PhD Programme: Mathematical Methods in Natural Sciences).

The aim of this paper is to present a parallel preconditioner for LDG discratization of (1) and to verify the method experimentally. In our design, we will follow the methods proposed by Baker, Brenner and Sung [5], who considered Poisson equation and showed the construction of the preconditioner for Poisson equation, as well as the results of Cai and Widlund [6], who designed a preconditioner for the second order elliptic equations (non-symmetric case) but discretized with the standard continuous Galerkin method.

The paper is organized as follows: Sect. 2 is dedicated to the presentation of local discontinuous Galerkin method. In Sect. 3 we construct the preconditioner for this problem. Section 4 concentrates on the implementation, and finally in Sect. 5 we show the results of numerical experiment, which confirm the theoretical results.

2 Construction of Discrete Problem

To construct the LDG method we use the mixed formulation. Let us rewrite (1) in the form of

$$\begin{cases} \boldsymbol{\sigma} = \nabla u & \text{in } \Omega, \\ -\nabla \cdot \boldsymbol{\sigma} + 2\boldsymbol{b} \cdot \nabla u + cu = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega. \end{cases} \tag{2}$$

Our goal is to find the solution of the system (1). Using (2) we find the solution u and also $\boldsymbol{\sigma}$, which stands for gradient of the solution. Usage of the mixed formulation causes that the dimension of considered problem is larger, but gives us much more information. Mixed formulation is usually used to construct the discontinuous Galerkin methods. It is also used in the implementation.

On the domain Ω we define a family of shape-regular triangulation denoted by $\mathcal{T}_h$, where h is a mesh parameter. We consider two finite element spaces:

$$\begin{aligned} V_h &= \left\{ v \in L^2(\Omega) : \quad v|_E \in P_1(E) \quad \forall E \in \mathcal{T}_h \right\}, \\ \Sigma_h &= \left\{ \boldsymbol{\tau} \in (L^2(\Omega))^2 : \quad \boldsymbol{\tau}|_E \in (P_1(E))^2 \quad \forall E \in \mathcal{T}_h \right\}, \end{aligned} \tag{3}$$

where $P_1(E)$ is the space of polynomial of degree less or equal to 1.

Let us denote by $\mathcal{E}_h$ the set of all edges of the triangulation elements. The set $\mathcal{E}_h$ can be splitted into two sets namely: $\mathcal{E}_h^{int}$ and $\mathcal{E}_h^{bd}$. The set $\mathcal{E}_h^{int}$ contains all edges from $\mathcal{E}_h$ which are located inside Ω and $\mathcal{E}_h^{bd}$ contains all edges from $\mathcal{E}_h$ which overlap with boundary $\partial\Omega$. Finite element spaces V_h and Σ_h contain discontinuous functions.

We use the average $\{\!\{\cdot\}\!\}$ and the jump $[\![\cdot]\!]$ of the function on the edge, cf. [5,7]. Both operators are defined for scalar and vector functions in the following way. Let q be a function from V_h, then

$$\begin{aligned} \{\!\{q\}\!\} &= \tfrac{1}{2}(q_1 + q_2) \quad & [\![q]\!] &= q_1\mathbf{n}_1 + q_2\mathbf{n}_2 \quad & \text{for } e \in \mathcal{E}_h^{int}, \\ \{\!\{q\}\!\} &= q & [\![q]\!] &= q\mathbf{n} & \text{for } e \in \mathcal{E}_h^{bd}, \end{aligned} \tag{4}$$

where $\boldsymbol{n}$ is the unit normal vector pointing outward Ω and $\boldsymbol{n}_1$ and $\boldsymbol{n}_1$ are the unit normal vector pointing outwards to the elements of the triangulation E_1

and E_2, respectively and $e = \overline{E}_1 \cap \overline{E}_2$. Let $\boldsymbol{\varphi}$ be a function from Σ_h, then

$$\begin{array}{lll} \{\!\{\boldsymbol{\varphi}\}\!\} = \frac{1}{2}(\boldsymbol{\varphi}_1 + \boldsymbol{\varphi}_2) & [\![\boldsymbol{\varphi}]\!] = \boldsymbol{\varphi}_1 \cdot \mathbf{n}_1 + \boldsymbol{\varphi}_2 \cdot \mathbf{n}_2 & \text{for } e \in \mathcal{E}_h^{int}, \\ \{\!\{\boldsymbol{\varphi}\}\!\} = \boldsymbol{\varphi} & [\![\boldsymbol{\varphi}]\!] = \boldsymbol{\varphi} \cdot \boldsymbol{n} & \text{for } e \in \mathcal{E}_h^{bd}. \end{array} \tag{5}$$

Using the mixed formulation we get the following system of equations

$$\begin{cases} \mathcal{H}(u_h, v) + \mathcal{G}(\boldsymbol{\sigma}_h, v) = \mathcal{F}(v), \\ -\mathcal{G}(\boldsymbol{\tau}, v) + \mathcal{D}(\boldsymbol{\sigma}_h, \boldsymbol{\tau}) = 0, \end{cases} \tag{6}$$

where $u_h \in V_h$, $\boldsymbol{\sigma}_h \in \Sigma_h$ are unknowns, $v \in V_h$, $\boldsymbol{\tau} \in \Sigma_h$ are the test functions and the bilinear forms $\mathcal{H}(\cdot,\cdot)$, $\mathcal{G}(\cdot,\cdot)$, $\mathcal{D}(\cdot,\cdot)$ and linear form $\mathcal{F}(\cdot)$ are defined by

$$\begin{aligned} \mathcal{H}(u_h, v) &= \sum_{e\in\mathcal{E}_h} \int_e \{\!\{u_h\}\!\}[\![v\boldsymbol{b}]\!] - \sum_{e\in\mathcal{E}_h} \int_e \{\!\{v\}\!\}[\![u_h\boldsymbol{b}]\!] - \sum_{e\in\mathcal{E}_h} \int_e \boldsymbol{\beta} \cdot [\![u_h]\!][\![v]\!] \cdot \boldsymbol{b} \\ &\quad + \sum_{E\in\mathcal{T}_h} \boldsymbol{b} \cdot \nabla u_h v - \sum_{E\in\mathcal{T}_h} \boldsymbol{b} \cdot \nabla v u_h + \sum_{e\in\mathcal{E}_h} \frac{\eta_e}{|e|} \int_e [\![u_h]\!][\![v]\!] + \int_\Omega c u_h v, \\ \mathcal{G}(\boldsymbol{\sigma}_h, v) &= \sum_{E\in\mathcal{T}_h} \int_E \nabla v \cdot \boldsymbol{\sigma}_h - \sum_{e\in\mathcal{E}_h} \int_e [\![v]\!] \cdot \{\!\{\boldsymbol{\sigma}_h\}\!\} - \sum_{e\in\mathcal{E}_h^{int}} \int_e \boldsymbol{\beta} \cdot [\![v]\!][\![\boldsymbol{\sigma}_h]\!], \\ \mathcal{D}(\boldsymbol{\sigma}_h, \boldsymbol{\tau}) &= \sum_{E\in\mathcal{T}_h} \boldsymbol{\sigma} \cdot \boldsymbol{\tau}, \\ \mathcal{F}(v) &= \sum_{E\in\mathcal{T}_h} \int_E f v, \end{aligned}$$

where $\boldsymbol{\beta}$ and η are the parameters of the method. It is assumed that $\boldsymbol{\beta}$ is bounded and constant on each edge and η is bounded, nonnegative and constant on each edge.

To eliminate the $\boldsymbol{\sigma}$, we have to define lifting operators, for more details see [5,7]. These operators appear in the bilinear form. Let us start from defining the local lifting operators

$$\begin{array}{llll} \int_\Omega r_e(\boldsymbol{\varphi}) \cdot \boldsymbol{\tau} = -\int_e \boldsymbol{\varphi} \cdot \{\!\{\boldsymbol{\tau}\}\!\} & e \in \mathcal{E}_h, & \forall \boldsymbol{\tau} \in \Sigma_h, & r_e : L^2(e) \to \Sigma_h, \\ \int_\Omega l_e(q) \cdot \boldsymbol{\tau} = -\int_e q [\![\boldsymbol{\tau}]\!] & e \in \mathcal{E}_h^{int}, & \forall \boldsymbol{\tau} \in \Sigma_h, & l_e : L^2(e) \to \Sigma_h. \end{array} \tag{7}$$

Global lifting operators are defined as a sum of the local lifting operators

$$r(\boldsymbol{\varphi}) = \sum_{e\in\mathcal{E}_h} r_e(\boldsymbol{\varphi}) \qquad \text{and} \qquad l(q) = \sum_{e\in\mathcal{E}_h^{int}} l_e(q). \tag{8}$$

After the elimination of $\boldsymbol{\sigma}$ we obtain the form of LDG discratization which depend only on u_h, then reads: find $u_h \in V_h$ such that

$$\mathcal{B}_h(u_h, v) = \mathcal{F}(v) \qquad \forall v \in V_h. \tag{9}$$

where

$$\mathcal{B}_h(u_h, v) = \mathcal{A}(u_h, v) + \mathcal{S}(u_h, v), \tag{10}$$

and

$$\mathcal{A}(u_h, v) = \sum_{E\in\mathcal{T}_h} \int_E \left(\nabla u_h + r([\![u_h]\!]) + l(\boldsymbol{\beta}\cdot[\![u_h]\!])\right)\cdot\left(\nabla v + r([\![v]\!]) + l(\boldsymbol{\beta}\cdot[\![v]\!])\right)$$
$$+ \sum_{e\in\mathcal{E}_h} \frac{\eta}{|e|}\int_e [\![u_h]\!]\cdot[\![v]\!] - \sum_{e\in\mathcal{E}_h^{int}} \int_e \boldsymbol{\beta}\cdot[\![u_h]\!][\![v]\!]\cdot\boldsymbol{b} + \int_\Omega c u_h v \tag{11}$$

and

$$\mathcal{S}(u_h, v) = \sum_{E\in\mathcal{T}_h}\int_E \boldsymbol{b}\cdot\nabla u_h v - \sum_{E\in\mathcal{T}_h}\int_E \boldsymbol{b}\cdot\nabla v u_h$$
$$+ \sum_{e\in\mathcal{E}_h^{int}}\int_e \{\!\{u_h\}\!\}[\![v\boldsymbol{b}]\!] - \sum_{e\in\mathcal{E}_h^{int}}\int_e [\![\boldsymbol{b}u_h]\!]\{\!\{v\}\!\}. \tag{12}$$

Bilinear form $\mathcal{A}(\cdot,\cdot)$ is symmetric and $\mathcal{S}(\cdot,\cdot)$ is skew-symmetric.

3 Additive Schwarz Method

I this Section we design an overlapping additive Schwarz method for (9), cf. [8]. We assume that $\mathcal{T}_h$ is obtained through a refinement procedure from the coarse, shape regular triangulation $\mathcal{T}_H$. The elements of $\mathcal{T}_H$ shall be denoted by Ω_i, $i = 1, .., N$, and will be referred to as “subdomains”. Thus, we can treat $\mathcal{T}_H$ as a decomposition of Ω into N subdomains of triangular shape.

Then we extend each subdomain to overlap its neighbours by an amount of δ, in such a way that the boundary of extended domain does not cut through any element of fine mesh. The new extended subdomain is named $\widehat{\Omega}_i$. The examples of coarse mesh $\mathcal{T}_H$ and the extended subdomain $\widehat{\Omega}_i$ are presented on the Fig. 1. Decomposition of Ω into subdomains fulfills the condition that every element $E \in \mathcal{T}_h$ should belong to at most N_s different $\widehat{\Omega}_i$ and N_s should be independent of N.

In order to define the additive Schwarz method, we define local finite element spaces related to these subdomains and the coarse finite element spaces related to the coarse triangulation. Let us start from the definition of local finite element spaces

$$V_i = \left\{ v \in V_h : \quad v|_{\Omega\setminus\widehat{\Omega}_i} = 0 \right\} \qquad \forall\, i = 1, .., N, \tag{13}$$

and then the coarse finite element space

$$V_0 = \{ v \in C_0(\Omega) : \quad v|_{\Omega_i} \in P_1(\Omega_i) \qquad i = 1, .., N \}\,. \tag{14}$$

Definition 1. *For $i = 0, .., N$, let us define projections $Q_i : V_h \to V_i$ and quasi-projections $P_i : V_h \to V_i$ by identifying*

$$\mathcal{B}_h(Q_i w, v) = \mathcal{B}_h(w, v),$$
$$\mathcal{A}(P_i w, v) = \mathcal{B}_h(w, v),$$

which holds for $v \in V_i$ and $w \in V_h$.

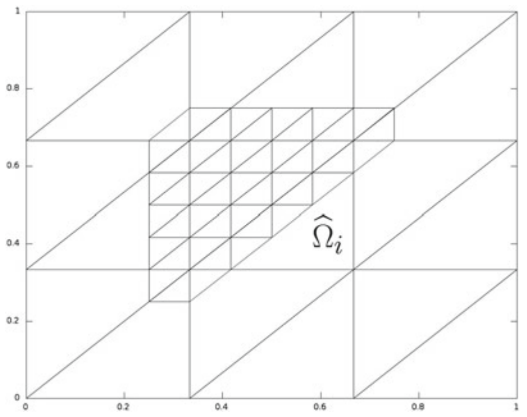


Fig. 1. An example of coarse triangulation $\mathcal{T}_H$ and the extended subdomain $\widehat{\Omega}_i$ with the elements of fine triangulation which belongs to $\widehat{\Omega}_i$. Size of the overlap presented above is equal to fine mesh size, i.e. $\delta = h$

Finally we define the operators $Q^{(1)} = \sum_{i=0}^{N} Q_i$ and $Q^{(2)} = \sum_{i=0}^{N} P_i$. Now problem (9) is replaced by

$$Q^{(i)} u_h = g_h^{(i)}, \qquad g_h^{(i)} \equiv \sum_{k=0}^{N} g_k^{(i)}, \tag{15}$$

where g_k can be calculated as the definition of projection (quasi-projection, respectively) and the right hand side of the original problem

$$\mathcal{B}_h(g_k^{(1)}, v) = f(v) \quad \textit{and} \quad \mathcal{A}_h(g_k^{(2)}, v) = f(v) \qquad \forall v \in V_k. \tag{16}$$

For the problems defined by (15) the following Theorem holds.

Theorem 1. *Let c (from (1) be non-negative function from L^∞, then the convergence speed of the GMRES with preconditioner $Q^{(1)}$ or $Q^{(2)}$ applied to problem (9) depends on the ratio $\frac{H}{\delta}$ and is independent of h.*

The proof of this theorem will appear in the forthcoming paper.

4 Implementation

Problem may be presented in two ways, i.e. standard (9) and mixed formulation (6). This leads to two different ways of analysis and problem implementation. First approach was defined in [5] and assumes the use of the bilinear form $\mathcal{B}_h(\cdot,\cdot)$. This approach is very useful for analysis of the problem. The other approach is discussed by Castillo and Sequeira in [10], where the authors solve the problem using the LDG method and mixed formulation. This approach causes that the system of linear equations becomes much bigger. Instead of 3 unknowns for each element of triangulation we get 9 unknowns. Using this approach, we obtain the solution of the problem u and the gradient of the solution $\boldsymbol{\sigma}$. Our system is much bigger, but the obtained solution gives us more information. Using the second approach we do not need to implement the lifting operators, hence it is simpler to implement.

Let us consider this problem from algebraic point of view. We identify each function $v_h \in V_h$ with the vector of the coefficients, $\overline{v} = (\alpha_1, \alpha_2, ..., \alpha_N)^T$, $v_h = \sum \alpha_i v_i$ and $\{v_i\}_{i=1,..,N}$ is a set of nodal functions of V_h. Now, the functions u_h and $\boldsymbol{\sigma}_h$ correspond to vectors $\overline{u}$ and $\overline{\sigma}$, respectively.

Taking nodal basis functions $\{v_i\}$ and $\{\boldsymbol{\sigma}_i\}$ from spaces V_h and Σ_h, respectively as a test functions in (6), we construct the matrices $\mathbb{H}$, $\mathbb{G}$ and $\mathbb{D}$ such that $\mathcal{H}(v_i, v_j) = \mathbb{H}_{i,j}$, $\mathcal{G}(\boldsymbol{\sigma}_i, v_j) = \mathbb{G}_{i,j}$ and $\mathcal{D}(\boldsymbol{\sigma}_i, \boldsymbol{\sigma}_j) = \mathbb{D}_{i,j}$. We obtain the algebraic system of equations

$$\begin{pmatrix} \mathbb{H} & \mathbb{G} \\ -\mathbb{G}^T & \mathbb{D} \end{pmatrix} \begin{pmatrix} \overline{u} \\ \overline{\sigma} \end{pmatrix} = \begin{pmatrix} \overline{f} \\ 0 \end{pmatrix}. \tag{17}$$

Matrix $\mathbb{D}$ is block diagonal, with 3×3 blocks, since $\mathbb{D}_{i,j} = \mathcal{D}(\boldsymbol{\varphi}_i, \boldsymbol{\varphi}_j) = \int_\Omega \boldsymbol{\varphi}_i \cdot \boldsymbol{\varphi}_j$. This matrix can be easily inverted, hence

$$-\mathbb{D}^{-1}\mathbb{G}^T \tag{18}$$

is cheap to compute and after eliminating $\overline{\sigma}$ from (6) we arrive of

$$\mathbb{B}_h \overline{u} \equiv (\mathbb{H} - \mathbb{G}\mathbb{D}^{-1}\mathbb{G}^T)\overline{u} = \overline{f} \tag{19}$$

To compute the matrix $\mathbb{B}_h$, we start from computing sparse matrices $\mathbb{H}$, $\mathbb{G}$ and $\mathbb{D}^{-1}$. Then, we get the matrix $\mathbb{B}_h$ from (19). The nonzero elements of $\mathbb{D}$, $\mathbb{G}$ and $\mathbb{H}$ are possible only for the bases function which has support inside the same element of triangulation or in two neighbouring elements of triangulation. Therefore, $\mathbb{B}_h$ remains sparse.

We defined two preconditioners in the previous Section. First one was constructed by using the projections and second one by using the quasi-projections. Now, we present the construction of first preconditioner.

We can define the restrictions from the finite element space V_h into the local and coarse spaces, i.e. $R_i : V_h \to V_i$ for $i = 1, .., N$, and let $\mathbb{R}_i$ be matrix corresponding to this restriction. The restriction operator R_0 is defined by the interpolant $R_0^T : V_0 \to V_h$. Operator R_0^T takes the vector from coarse space and gives a vector from fine space that determines the corresponding coarse function.

Rewriting the system (15) in the algebraic form with preconditioner (for $Q^{(1)}$) we get the following system of equations

$$\left(\sum_{i=0}^{N} \mathbb{R}_i^T \mathbb{B}_i^{-1} \mathbb{R}_i \right) \mathbb{B}\overline{u} = \left(\sum_{i=0}^{N} \mathbb{R}_i^T \mathbb{B}_i^{-1} \mathbb{R}_i \right) \overline{f}, \tag{20}$$

and the the matrix $\sum_{i=0}^{N} \mathbb{R}_i^T \mathbb{B}_i^{-1} \mathbb{R}_i$ is a parallel preconditioner of the system, because each $\mathbb{B}_i = \mathbb{R}_i \mathbb{B}_h \mathbb{R}_i^T$ can be applied independently to the residual vector.

Similarly, using the quasi-projection for construction of the preconditioner (for $Q^{(2)}$), we get the following system of equations

$$\left(\sum_{i=0}^{N} \mathbb{R}_i^T \mathbb{A}_i^{-1} \mathbb{R}_i\right) \mathbb{B}\overline{u} = \left(\sum_{i=0}^{N} \mathbb{R}_i^T \mathbb{A}_i^{-1} \mathbb{R}_i\right) \overline{f}. \tag{21}$$

5 Numerical Experiments

Let Ω be a square $(0,1) \times (0,1)$. We consider two test problems, cf. [6].

Problem 1. Let us consider the symmetric and indefinite Helmholtz equation

$$\begin{cases} -\Delta u - \mu u = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega. \end{cases} \tag{22}$$

Problem 2. Let us consider a nonsymmetric and indefinite problem

$$\begin{cases} -\Delta u - \nu(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}) - \mu u = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega. \end{cases} \tag{23}$$

The aim of these tests is to assess the efficiency of both preconditioners proposed in Sect. 4. All of the tests are done for the right hand side corresponding to the exact solution $u = xe^{xy} \sin \pi x \sin \pi y$. We use LDG parameters $\boldsymbol{\beta} = (1,1)^T$ and $\eta = 1$.

Each of these problems is solved in three ways, i.e. without preconditioner and with one of the preconditioner presented in the previous Section. In all tests we stop the GMRES method as soon as the ratio $\frac{\|r_i\|_h}{\|r_0\|_h}$ is less then 10^{-3}, where the norm $\|\cdot\|_h$ is defined by

$$\|v\|_h^2 = \sum_{E \in \mathcal{T}_h} \int_E |\nabla v|^2 + \sum_{e \in \mathcal{E}_h} \frac{1}{|e|} \int_e |[\![v]\!]|^2, \tag{24}$$

(cf. [5]) or after 400 iterations.

Tables 1–6 present the results of the numerical tests, i.e. the number of iteration needed to achieve the proper ratio of the norms of residual vectors. We show the number of iterations for the GMRES method without preconditioner, named by no preconditioner in the tables, and the number of iterations for the GMRES method with preconditioners ($Q^{(1)}$ and $Q^{(2)}$). In the numerical tests we take the different values of the mesh sizes and overlaps. For Problems 1 and 2 we analyze dependency of iterations numbers with the mesh sizes h and H and with the size of the overlap δ.

In Tables 1 and 4 we present the result for constant ratio $\frac{H}{\delta}$ in two cases. In the experiment we change parameters h, H and δ in such way that ratio $\frac{H}{\delta}$

Table 1. Results for the Problem 1 with $\mu = -3\pi^2$.

h^{-1}	$\frac{H}{h}$	$\frac{\delta}{h}$	$\frac{H}{\delta}$	no preconditioner	$Q^{(1)}$	$Q^{(2)}$
15	5	2	$\frac{5}{2}$	166	12	12
30	10	4	$\frac{5}{2}$	325	12	13
45	15	6	$\frac{5}{2}$	401	12	13
60	20	8	$\frac{5}{2}$	401	12	14
15	3	1	3	166	8	9
30	6	2	3	325	9	9
45	9	3	3	401	9	9
60	12	4	3	401	9	9

Table 2. Results for the Problem 1 with $\mu = -3\pi^2$. Dependency on overlap δ.

h^{-1}	$\frac{H}{h}$	$\frac{\delta}{h}$	$\frac{H}{\delta}$	no preconditioner	$Q^{(1)}$	$Q^{(2)}$
80	10	1	10	401	10	10
80	10	2	5	401	9	9
80	10	3	$\frac{10}{3}$	401	9	9
80	10	4	$\frac{5}{2}$	401	10	10
80	10	5	2	401	11	11
80	10	6	$\frac{5}{3}$	401	12	12
80	10	7	$\frac{10}{7}$	401	12	13
80	10	8	$\frac{10}{8}$	401	12	12

Table 3. Results for the Problem 1 with $\mu = -3\pi^2$. Dependency on coarse mesh size H.

h^{-1}	$\frac{H}{h}$	$\frac{\delta}{h}$	$\frac{H}{\delta}$	no preconditioner	$Q^{(1)}$	$Q^{(2)}$
80	2	2	1	401	14	14
80	4	2	2	401	10	10
80	5	2	$\frac{5}{2}$	401	9	9
80	8	2	4	401	9	9
80	10	2	5	401	9	9
80	16	2	8	401	10	10
80	20	2	10	401	15	15

is equal to $\frac{5}{2}$ (first case) or 3 (second case). The fluctuation of the iterations number of GMRES method with usage of preconditioner is almost constant for each problem. Only using the preconditioner $Q^{(2)}$ we can observe slow growth of this number with the mesh sizes h is getting smaller.

In Tables 2 and 5 we present the results for constant mesh sizes h and H. In all tests $h = \frac{1}{80}$ and $H = 8h$. Size of overlap δ is changing form h to $8h$. As we can see small and big overlap implies higher number of iterations needed to solve the preconditioned system.

In Tables 3 and 6 we present the results for constant mesh size h and overlap size δ. In all tests $h = \frac{1}{80}$ and $\delta = 2h$. The coarse mesh size H is changing from $2h$

Table 4. Results for the Problem 2 with $\mu = -3\pi^2$ and $\nu = -3\pi$.

h^{-1}	$\frac{H}{h}$	$\frac{\delta}{h}$	$\frac{H}{\delta}$	no preconditioner	$Q^{(1)}$	$Q^{(2)}$
15	5	2	$\frac{5}{2}$	145	12	17
30	10	4	$\frac{5}{2}$	298	12	18
45	15	6	$\frac{5}{2}$	401	12	18
60	20	8	$\frac{5}{2}$	401	12	18
15	3	1	3	145	13	16
30	6	2	3	298	14	17
45	9	3	3	401	14	17
60	12	4	3	401	14	18

Table 5. Results for the Problem 2 with $\mu = -3\pi^2$ and $\nu = -3\pi$. Dependency on overlap δ.

h^{-1}	$\frac{H}{h}$	$\frac{\delta}{h}$	$\frac{H}{\delta}$	no preconditioner	$Q^{(1)}$	$Q^{(2)}$
80	10	1	10	401	16	17
80	10	2	5	401	15	17
80	10	3	$\frac{10}{3}$	401	14	15
80	10	4	$\frac{5}{2}$	401	13	15
80	10	5	2	401	15	16
80	10	6	$\frac{5}{3}$	401	16	19
80	10	7	$\frac{10}{7}$	401	16	20
80	10	8	$\frac{10}{8}$	401	15	19

Table 6. Results for the Problem 2 with $\mu = -3\pi^2$ and $\nu = -3\pi$. Dependency on coarse mesh size H.

h^{-1}	$\frac{H}{h}$	$\frac{\delta}{h}$	$\frac{H}{\delta}$	no preconditioner	$Q^{(1)}$	$Q^{(2)}$
80	2	2	1	401	22	22
80	4	2	2	401	13	13
80	5	2	$\frac{5}{2}$	401	11	12
80	8	2	4	401	14	15
80	10	2	5	401	15	17
80	16	2	8	401	18	21
80	20	2	10	401	21	24

to $20h$. If we omit the first test in these two tables, we can see that the number of GMRES iterations in preconditioned system is growing with the ratio $\frac{H}{\delta}$.

As we can see the number of iterations in the algorithms with precondtioner is independent of the mesh size h. The same results were presented in [6], hence using the preconditioners constructed by Additive Schwarz method for continuous Galerkin and local discontinuous methods makes the condition of the problem independent of the mesh size h. Moreover, the number of iterations needed for solving the problem with the use of preconditioner is lower for $Q^{(1)}$

than for $Q^{(2)}$, hence solving the inexact problem gives us worse results. This result is different than the one presented by Cai and Widlund in [6], where the precondtioner constructed with usage the quasi-projections needs lower number of iterations. Results of the numerical tests confirm the results presented in the Theorem 1, proof of which will appear elsewhere. Tests show that even in the case of negative function c the condition of the preconditioner is independent of mesh size h.

References

1. Reed, W.H., Hill, T.R.: Triangular mesh methods for the neutron transport equation. Technical report LA-UR-73-479, Los Alamos Scientific Laboratory (1973)
2. Cockburn, B., Shu, C.: The local discontinuous Galerkin method for time-dependent convection-diffusion systems. SIAM J. Numer. Anal. **35**(6), 2440–2463 (1998)
3. Cockburn, B., Kanschat, G., Perugia, I., Schötzau, D.: Superconvergence of the local discontinuous Galerkin method for elliptic problems on cartesian grids. SIAM J. Numer. Anal. **39**, 264–285 (2001)
4. Castillo, P., Cockburn, B., Perugia, I., Schötzau, D.: An a priori error analysis of the local discontinuous Galerkin method for elliptic problems. SIAM J. Numer. Anal. **38**(5), 1676–1706 (2000)
5. Barker, A., Brenner, S., Sung, L.Y.: Overlapping Schwarz domain decomposition preconditioners for the local discontinuous Galerkin method for elliptic problem. J. Numer. Math. 1–25 (2011)
6. Cai, X., Widlund, O.: Domain decomposition algorithms for indefinite elliptic problems. SIAM J. Sci. Stat. Comput. **13**(1), 243–258 (1992)
7. Arnold, D., Brezzi, F., Cockburn, B., Marini, L.: Unified analysis of discontinuous Galerkin methods for elliptic problems. SIAM J. Numer. Anal. **39**(5), 1749–1779 (2002)
8. Toselli, A., Widlund, O.: Domain Decomposition Methods - Algorithms and Theory. Springer, Berlin (2005)
9. Brenner, S.: Two-level additive Schwarz preconditioners for nonconforming finite element methods. Math. Comp. **65**(215), 897–921 (1996)
10. Castillo, P., Sequeira, F.: Computational aspects of the local discontinuous Galerkin method on unstructured grids in three dimensions. Math. Comput. Model. **57**(910), 2279–2288 (2013)

Fast Numerical Method for 2D Initial-Boundary Value Problems for the Boltzmann Equation

Alexei Heintz[1] and Piotr Kowalczyk[2](✉)

[1] Department of Mathematics, Chalmers University of Technology and Göteborg University, SE-412 96 Göteborg, Sweden
heintz@chalmers.se

[2] Institute of Applied Mathematics and Mechanics, University of Warsaw, Banacha 2, 02-097 Warszawa, Poland
pkowal@mimuw.edu.pl

Abstract. We present a new numerical scheme for the initial-boundary value problem for the Boltzmann equation in two-dimensional physical space. It is based on a splitting procedure in which the collision equation is solved using the adaptive algorithm for the computation of the full three-dimensional Boltzmann collision operator on non-uniform velocity grids introduced in the previous paper by the authors. The computation of the collision operator is performed in parallel for every physical grid cell. For the two-dimensional transport equation we use a second order finite volume method. The numerical example showing the effectiveness of our method is given.

Keywords: Boltzmann equation · Numerical methods · Non-uniform grids

1 Introduction

In thisaut]Heintz, Alexeiaut]Kowalczyk, Piotr paper we present a deterministic method for numerical solution of the two-dimensional initial-boundary value problems for the classical Boltzmann equation [8]

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f = Q(f, f) \tag{1}$$

which is the main object in the kinetic theory of rarefied gases. In (1) the function $f := f(t, \mathbf{x}, \mathbf{v})$, $f : \mathbb{R}_+ \times \Omega_x \times \mathbb{R}^3 \to \mathbb{R}_+$ is a probability density function of the gas molecules which at time t and at the point $\mathbf{x} \in \Omega_x \subset \mathbb{R}^d$ move with the velocity $\mathbf{v}$. The collision integral $Q(f, f)$ describes changes in f due to collisions between molecules. The Boltzmann equation (1) is equipped with the initial $f(0, \mathbf{x}, \mathbf{v}) = f_0(\mathbf{x}, \mathbf{v})$ and boundary $f(t, \mathbf{x}, \mathbf{v}) = g(t, \mathbf{x}, \mathbf{v})$, $\mathbf{x} \in \partial\Omega_x$, conditions for some suitably chosen functions f_0 and g.

The numerical solution of the Boltzmann equation is very difficult because of the nonlinearity of the collision integral, the large number of independent variables and the complicated integration over a five-dimensional cone in the six-dimensional space of pre- and post-collisional velocities. Due to this numerical complexity many practical computations of the Boltzmann equation are based on various probabilistic Monte Carlo methods, see e.g. [4,17,22]. However these methods have limited accuracy. Development of deterministic methods for the Boltzmann equation is usually motivated (see [18,21]) by the desire of higher precision results in situations when probabilistic methods are not effective enough.

The discrete velocity models (DVM) defined on a uniform cubic grid of velocities were the first deterministic approximations to the Boltzmann equation. Several different ideas led to various types of DVM all satisfying exact conservation laws in the discrete form [7,12,19,23]. All mentioned DVM methods have high computational cost n^7, where n denotes the number of points along one coordinate direction in the uniform velocity grid.

The Kyoto group in kinetic theory has developed a family of finite difference methods for the Boltzmann equation [15,18]. These computations demonstrate precise results but they are very time and memory consuming.

Another class of deterministic methods is based on Fourier spectral technique. A general spectral method based on the restriction of the Boltzmann equation to a finite domain and on the representation of the solution by Fourier series was first derived in [20] and developed further in [21]. The method was also applied to spatially non-homogeneous Boltzmann equation [10,11]. Independently in [5,6] the authors constructed fast algorithms based on Fourier transform approximation. In papers [9,16] a new important idea of semi-discretization was introduced. This idea led to a great reduction of the computational cost up to the order $N^k \log_2 N$ for velocity dimension k and N the number of Fourier modes along one coordinate direction, without losing spectral accuracy.

In our previous paper [13] we addressed a problem of approximating solutions to the homogeneous Boltzmann equation discontinuous with respect to the velocity variable. We combined two approaches: an adaptive velocity grid to resolve discontinuities and a spectral method to approximate smooth terms in the equation: the collision frequency and the gain term in the collision operator. This idea is based on the fact that the gain term $Q^+(f,f)$ in the collision operator has certain smoothing properties and is actually smooth even for a discontinuous f. Collision frequency $q^-(f)$ is also a smooth function because it is a convolution of f with a regular function.

The goal of the present paper is to show the effectiveness of the approach of [13] in the case of spatially two-dimensional problems. The outline of the paper is as follows. In Sect. 2 we present the Boltzmann equation and the Fourier representation. In Sect. 3 we give the numerical framework of the paper. Subsection 3.1 describes the classical splitting scheme. In Subsect. 3.2 we introduce the adaptive approximation of the distribution function in velocity space with the spectral approximation of smooth terms in the collision operator, which is based on the USFFT scheme and the standard spectral method (see [2,21]). In Subsect. 3.3 we describe the numerical scheme used for the collisionless transport step of the splitting method. In Sect. 4 we present the numerical example.

2 Boltzmann Equation

In this paper we consider the gas model of “hard sphere” molecules [3]. In this model the collision operator $Q(f,f)$ in (1) is defined as follows

$$Q(f,f)(\mathbf{v}) := \int_{\mathbb{R}^3}\int_{S^2} |\mathbf{v}-\mathbf{w}| \left[f(\mathbf{v}')f(\mathbf{w}') - f(\mathbf{v})f(\mathbf{w})\right] d\omega\, d\mathbf{w}, \tag{2}$$

where $\mathbf{v}$, $\mathbf{w}$ are the velocities of the particles before the collision and $\mathbf{v}'$, $\mathbf{w}'$ — the velocities of the particles after collision given by

$$\mathbf{v}' := \frac{1}{2}\left(\mathbf{v}+\mathbf{w}+|\mathbf{v}-\mathbf{w}|\omega\right), \qquad \mathbf{w}' := \frac{1}{2}\left(\mathbf{v}+\mathbf{w}-|\mathbf{v}-\mathbf{w}|\omega\right).$$

The collision operator (2) can be decomposed into the *gain* and the *loss* parts

$$Q(f,f)(\mathbf{v}) = Q^+(f,f)(\mathbf{v}) - Q^-(f,f)(\mathbf{v}),$$

where

$$Q^+(f,f)(\mathbf{v}) = \int_{\mathbb{R}^3}\int_{S^2} |\mathbf{u}| f(\mathbf{v} - \frac{1}{2}(\mathbf{u} - |\mathbf{u}|\omega)) f(\mathbf{v} - \frac{1}{2}(\mathbf{u} + |\mathbf{u}|\omega)\, d\omega\, d\mathbf{u} \tag{3}$$

and

$$Q^-(f,f)(\mathbf{v}) = f(\mathbf{v}) q^-(f)(\mathbf{v})$$

with q^- denoting the *collision frequency* term

$$q^-(f)(\mathbf{v}) = \int_{\mathbb{R}^3} f(\mathbf{v}-\mathbf{u}) \int_{S^2} |\mathbf{u}|\, d\omega\, d\mathbf{u}. \tag{4}$$

We have changed the variables $\mathbf{u} = \mathbf{v} - \mathbf{w}$ in (2) to get the above formulations.

We will use the following form of the Fourier transform

$$\mathcal{F}_v(\mathbf{m})[f] := \hat{f}_{\mathbf{m}} = \int_{\mathbb{R}^3} f(\mathbf{v}) e^{2\pi\imath(\mathbf{v},\mathbf{m})}\, d\mathbf{v} \tag{5}$$

and the inverse Fourier transform

$$\mathcal{F}_m^{-1}(\mathbf{v})[\hat{f}] := f(\mathbf{v}) = \int_{\mathbb{R}^3} \hat{f}_{\mathbf{m}} e^{-2\pi\imath(\mathbf{m},\mathbf{v})}\, d\mathbf{m}. \tag{6}$$

One can reformulate the gain and collision frequency terms using the Fourier transform:

$$Q^+(f,f)(\mathbf{v}) = \mathcal{F}_l^{-1}(\mathbf{v})\mathcal{F}_m^{-1}(\mathbf{v})\left[\hat{f}_{\mathbf{l}}\hat{f}_{\mathbf{m}}\hat{B}(\mathbf{l},\mathbf{m})\right],$$
$$q^-(f)(\mathbf{v}) = \mathcal{F}_m^{-1}(\mathbf{v})\left[\hat{f}_{\mathbf{m}}\hat{B}(\mathbf{m},\mathbf{m})\right],$$

where

$$\hat{B}(\mathbf{l},\mathbf{m}) = \int_{\mathbb{R}^3}\int_{S^2} |\mathbf{u}| e^{2\pi\imath(\frac{\mathbf{l}+\mathbf{m}}{2},\mathbf{u})} e^{2\pi\imath|\mathbf{u}|(\frac{\mathbf{m}-\mathbf{l}}{2},\omega)}\, d\omega d\mathbf{u}.$$

The kernel $\hat{B}(\mathbf{l},\mathbf{m})$ is a distribution, hence in order to use it in practical computations we have to regularize it. We will choose in a proper way the constant $R > 0$ and write the regularized kernel as

$$\hat{B}_R(\mathbf{l},\mathbf{m}) = \int_{\mathcal{B}(0,R)}\int_{S^2} |\mathbf{u}| e^{2\pi\imath(\frac{\mathbf{l}+\mathbf{m}}{2},\mathbf{u})} e^{2\pi\imath|\mathbf{u}|(\frac{\mathbf{m}-\mathbf{l}}{2},\omega)}\, d\omega d\mathbf{u}. \tag{7}$$

We denote by $Q_R^+(f,f)$ and $q_R^-(f)$ the gain and collision frequency terms with the regularized kernel $\hat{B}_R$. The kernel $\hat{B}_R$ was computed analytically in [21] in particular for hard sphere gas.

3 Numerical Algorithm

3.1 Splitting Method

The numerical procedure most commonly used for solving the non-stationary Boltzmann equation consists in splitting the Boltzmann equation on each time step into the *transport* step with the equation for collisionless gas and the *relaxation* step with the spatially homogeneous Boltzmann equation.

We divide the time interval $[0,T]$ onto N equal subintervals of length $\Delta t = T/N$. Assume that the approximate value f^n of the distribution function at the time $n\Delta t$ has been computed. Then f^{n+1}, the value of the distribution function at the time $(n+1)\Delta t$, is obtained in two steps:

1. the transport equation

$$\begin{aligned} \frac{\partial f^*}{\partial t} + \mathbf{v}\cdot\nabla_{\mathbf{x}} f = 0 \quad \text{for} \quad t \in (n\Delta t, (n+1)\Delta t]\,, \\ f^*(k\Delta t) = f^n\,, \end{aligned} \tag{8}$$

2. the relaxation equation

$$\begin{aligned} \frac{\partial f^{**}}{\partial t} = Q(f^{**}, f^{**}) \quad \text{for} \quad t \in (n\Delta t, (n+1)\Delta t]\,, \\ f^{**}(n\Delta t) = f^*((n+1)\Delta t)\,. \end{aligned} \tag{9}$$

Finally, using the solution of (9) one sets

$$f^{n+1} = f^{**}((n+1)\Delta t)\,.$$

The boundary conditions appropriate for the particular problem have to be taken into account when solving the free flow stage (8) of the splitting procedure.

3.2 Relaxation Equation

Below we describe briefly the main points of the algorithm for the homogeneous Boltzmann equation. For the detailed description we refer to [13].

For the numerical treatment of the Boltzmann equation we restrict the function f to a bounded domain Ω_v. To discretize the velocity space we build a nonuniform adaptive velocity grid $\mathcal{G}_v \subset \Omega_v$ with N_v denoting the number of points in $\mathcal{G}_v$ which allows to resolve discontinuities and high gradients in f.

The velocity grid $\mathcal{G}_v$ is created in parallel with the computation of the values of the solution at each time step and for each space point. We compute first the values of the solution at the vertices of cubic cells on some initial uniform coarse grid. We fix in advance a desired resolution (minimal grid step) in velocity space and desired maximal variation of solution within the cells exceeding this resolution. Then choosing cubes where variation of the solution is larger than the desired maximal variation we divide these cells into eight similar smaller ones, compute the values of the solution at the vertices of new cubes and then continue the subdivision in the same way. The subdivision is stopped when the minimal size of the cells is reached. In such a way we generate more grid points in parts of Ω_v where the function f changes rapidly and less in where the function is almost constant.

For smooth terms in the equation: the gain term $Q^+(f, f)$ and the collision frequency $q^-(f)$ we use a spectral representation, that is the projection onto the basis of high (5-th or 7-th) order B-splines. This projection is a component of the *Unequally Spaced Fast Fourier Transform* (USFFT) algorithm by Beylkin [2] that we use for the computation of expressions (3) and (4) for $Q^+(f, f)$ and $q^-(f)$.

Using a non-uniform approximation $\bar{f}$ for f on the grid $\mathcal{G}_v$ we get the following discrete in velocity variable approximation for the Fourier transform of f (5) (we keep here the same notation for the Fourier transform of f and its approximation)

$$\hat{f}_{\mathbf{m}} = C_{\mathbf{m}} \sum_j \bar{F}_j \, e^{2\pi\imath(\mathbf{v}_j, \mathbf{m})},$$

where $\bar{F}_j$ is a sum of contributions to the node $\mathbf{v}_j$ of the approximations $\bar{f}_K$ on cube $K \in \mathcal{G}_v$. We adopt the USFFT algorithm here, because the standard Fast Fourier Transform (FFT) algorithm commonly used for the fast computation of trigonometric sums cannot be used here since the points $\mathbf{v}_j$ in the velocity grid $\mathcal{G}_v$ are not equidistant. The cost of the USFFT algorithm in our case can be estimated as $\mathrm{O}(N_v) + \mathrm{O}(N^3 \log N)$, where $N = 4M$ with $2M$ denoting the number of Fourier modes in one direction. A similar procedure is used to calculate the inverse Fourier transform (6), uniformly discretized in Fourier domain, with the cost the same as above.

The relaxation step of the splitting method is carried out using the standard semi-implicit Euler scheme

$$f^{n+1} = \frac{f^n + \Delta t \, Q_R^+(f^n, f^n)}{1 + \Delta t \, q_R^-(f^n)}, \tag{10}$$

where $f^n = f(t_n)$ denotes the solution f at time t_n. The equation (10) is solved for every $\mathbf{v} \in \mathcal{G}_v$ and $\mathbf{x} \in \Omega_x$.

The kernel $\hat{B}_R$ (7) depends only on $|\mathbf{l}+\mathbf{m}|$ and $|\mathbf{l}-\mathbf{m}|$ (see [21]). Thus we define $T(|\mathbf{p}|,|\mathbf{q}|) := \hat{B}_R(\mathbf{l},\mathbf{m})$ with $\mathbf{p} = \mathbf{l}+\mathbf{m}$ and $\mathbf{q} = \mathbf{l}-\mathbf{m}$ and using the inverse Fourier transform we discretize the regularized gain term $Q_R^+(f,f)$ (cf. (7)) as follows

$$Q_R^+(f,f)(\mathbf{v}) = \sum_{\mathbf{p}} F(\mathbf{p}) e^{-2\pi\imath(\mathbf{p},\mathbf{v})},$$

where $F(\mathbf{p}) = \sum_{\mathbf{q}} \hat{f}_{\frac{\mathbf{p}+\mathbf{q}}{2}} \hat{f}_{\frac{\mathbf{p}-\mathbf{q}}{2}} T(|\mathbf{p}|,|\mathbf{q}|)$. Here we calculate $F(\mathbf{p})$ for every $\mathbf{p}$ with a cost $O(N^6)$ and do the inverse USFFT to calculate $Q_R^+(f,f)(\mathbf{v})$. Similarly we discretize the loss term $Q_R^-(f,f)$.

The total computational cost of the numerical approximation of the collision operator is $\mathrm{O}(N_v) + \mathrm{O}(N^6 \log N)$, where the first part includes approximation of the solution on the adaptive velocity grid Ω_v with N_v points and the second part depends on the moderately large number N of Fourier modes taken along one coordinate direction for the approximation of the collision operator. The velocity grids for each point in physical space are independent, hence the computations can be performed in parallel. This gives nearly linear performance gain as the number of processors increases up to the size of the physical space grid. This parallelization gives very high performance gain, because this step of the splitting procedure has the largest computational complexity in our algorithm.

The FFT scheme conserves exactly only mass — momentum and energy are conserved only up to the precision of the method. Hence we use the correction technique proposed in [1] for enforcing exact conservation property.

3.3 Transport Equation

The transport equation (8) is solved numerically using the second order finite volume scheme for the hyperbolic advection equation (see e.g. [7,14]). For completeness, we give a brief description of the scheme we use in our algorithm.

Let $\mathcal{M}$ denote the uniform partitioning of the domain Ω_x into rectangles. Let $M \in \mathcal{M}$ be a control cell having the vertices A, B, C, D. We denote by $\mathbf{n}_{AB}$ the unit outward normal to the side AB, by $|AB|$ its length, by S_{AB} the midpoint of AB, and finally by M_{AB} the cell adjacent to the side AB. Similar notation is used for each of the other sides. Moreover, $|M|$ stands for the area of the cell M and S_M for its center.

In what follows we suppress in the notation the explicit dependence of f on the velocity variable. Let f_M^n be an approximation of

$$\frac{1}{|M|} \int_M f(t_n, \mathbf{x}) d\mathbf{x} \tag{11}$$

and let g be a function defined on Ω_x such that its restriction g_M to the cell M is defined as

$$g_M(\mathbf{x}) = f_M^n + (\nabla_{\mathbf{x}} f)_M^n (\mathbf{x} - S_M) \quad \text{for} \quad \mathbf{x} \in M\,, \tag{12}$$

where $(\nabla_{\mathbf{x}} f)_M^n$ denotes an approximation of the gradient of f on the cell M.

Let q_{AB} be the flux through the side AB defined as follows

$$q_{AB} = \begin{cases} \mathbf{v} \cdot \mathbf{n}_{AB}\, g_{AB}^{in}\, |AB|, & \text{if } \mathbf{v} \cdot \mathbf{n}_{AB} \geq 0\,, \\ \mathbf{v} \cdot \mathbf{n}_{AB}\, g_{AB}^{out}\, |AB|, & \text{if } \mathbf{v} \cdot \mathbf{n}_{AB} < 0\,, \end{cases} \tag{13}$$

where

$$\begin{aligned} g_{AB}^{in} &= \lim\nolimits_{M \ni \mathbf{x} \to S_{AB}} g(\mathbf{x})\,, \\ g_{AB}^{out} &= \lim\nolimits_{M \not\ni \mathbf{x} \to S_{AB}} g(\mathbf{x})\,. \end{aligned}$$

Likewise, we define the fluxes through the other sides of the cell M.

The second order finite volume scheme is defined as follows

$$f_M^{n+1} = f_M^n - \frac{\Delta t}{|M|}(q_{AB} + q_{BC} + q_{CD} + q_{DA})\,.$$

The scheme preserves positivity and is stable under the condition (proof in [7])

$$\max_{\mathbf{v} \in \Omega_v} \max_{M \in \mathcal{M}} \frac{\Delta t}{|M|} \max_{XY} (\max(\mathbf{v} \cdot \mathbf{n}_{XY}, 0)|XY|) \leq \frac{1}{4}\,. \tag{14}$$

We use the adaptive non-uniform grid for the velocity space. Hence the velocity grids corresponding to different space cells might not coincide. To overcome this problem the required values of the distribution function in the neighboring space cells are calculated in the velocity points using a piecewise linear approximation within the cube of the velocity grid.

4 Numerical Example

In this section we present a results of numerical computations for a shock wave reflection problem showing the effectiveness of our method in 2D case.

We investigate a problem of a planar shock wave moving in a tube with a rectangular obstacle inside (see Fig. 1). We obtain the incoming shock wave as follows. Initially the domain Ω_x is divided into two parts Ω_1 and Ω_2 and in each part the gas is in equilibrium. Hence the initial distribution function is given by

$$f_0(\mathbf{x}, \mathbf{v}) = \begin{cases} M(\rho_1, 0, T_1, \mathbf{v}), & \text{if } \mathbf{x} \in \Omega_1, \\ M(\rho_2, \mathbf{u}, T_2, \mathbf{v}), & \text{if } \mathbf{x} \in \Omega_2, \end{cases}$$

where $M(\rho, \mathbf{u}, T, \mathbf{v}) = \frac{\rho}{(2\pi RT)^{3/2}} \exp\big(-\frac{(\mathbf{v}-\mathbf{u})^2}{2RT}\big)$ and T_1, ρ_1, T_2, ρ_2 are the gas temperature and the density before and behind the shock, respectively, chosen to satisfy the Rankine-Hugoniot relations for the shock with Mach number $M = 2.8$.

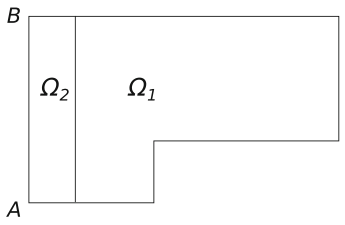


Fig. 1. Domain of a shock wave reflection problem

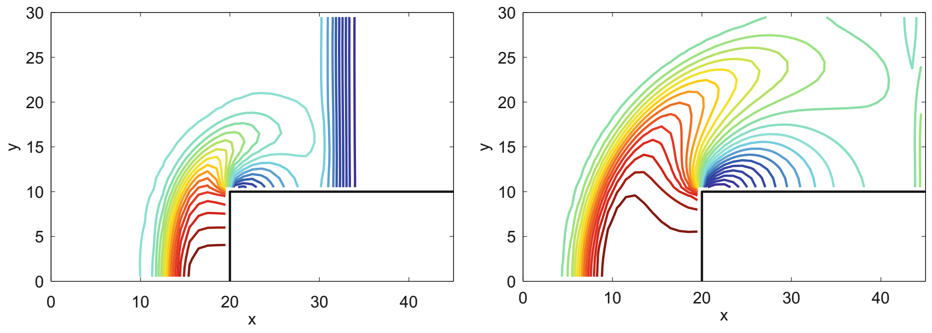


Fig. 2. Shock wave: density contour lines for $t = 300$ (left) and $t = 440$ (right)

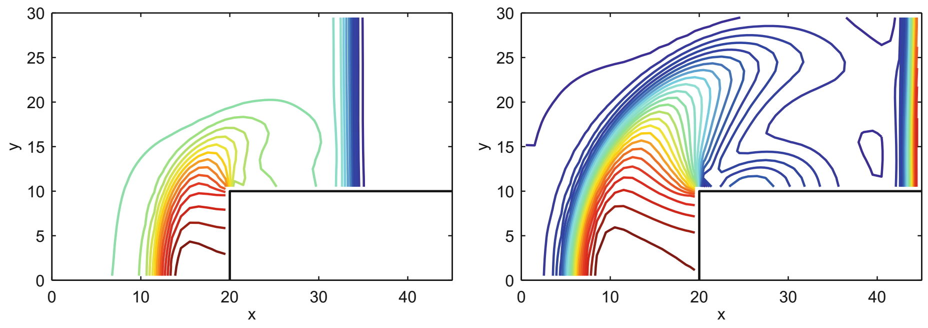


Fig. 3. Shock wave: temperature contour lines for $t = 300$ (left) and $t = 440$ (right)

The interaction of the gas with the boundary Ω_x is modeled using the specular reflection

$$f(t,\mathbf{x},\mathbf{v}) = f(t,\mathbf{x},\mathbf{v} - 2(\mathbf{n}(\mathbf{x})\cdot\mathbf{v})\mathbf{n}(\mathbf{x})) \text{ for } \mathbf{x} \in \partial\Omega_x \backslash \overline{AB} \text{ and } \mathbf{v}\cdot\mathbf{n}(\mathbf{x}) > 0,$$

where $\mathbf{n}(\mathbf{x})$ denotes the inner unit normal to the boundary $\partial\Omega_x$ at the point $\mathbf{x}$. This boundary condition is used on all the boundary except the left hand side (the interval $\overline{AB}$), where the constant inflow of the gas is given

$$f(t,\mathbf{x},\mathbf{v}) = M(\rho_2,\mathbf{u},T_2,\mathbf{v}) \text{ for } \mathbf{x} \in \overline{AB} \text{ and } \mathbf{v}\cdot\mathbf{n}(\mathbf{x}) > 0.$$

The numerical calculations presented here were performed for the domain Ω_x partitioned into rectangles with the sides $\Delta x = \Delta y = 1.0$ and the time step was $\Delta t = 0.035714$, chosen to satisfy (14). In the calculations we used the Boltzmann equation in a dimensionless form and assumed the Knudsen number $Kn = 1$. Because of the high dimensionality of the considered problems (2D in space and 3D in velocity), calculating experimentally the approximation order of the scheme requires more powerful computer than the authors used and a better implementation of the algorithm. Hence we note here only that the calculations of the presented example took 26 hours using a 2-core processor.

In Figs. 2 and 3 we show the density and temperature contour lines for time steps $t = 300$ and $t = 440$.

For time step $t = 300$ both the incoming shock wave and the shock reflected from the obstacle are observed. At time step $t = 440$ the density shock wave has reached the wall and we observe only the shock reflected from the obstacle, whereas for the temperature shock wave we observe also the shock reflecting from the wall. In all figures the incoming shock wave is moving from the left to the right.

5 Conclusions

In this paper we present a new numerical method for the solution of the two-dimensional initial-boundary value problems for the Boltzmann equation.

The method uses a splitting scheme to decouple the transport part and the collision part of the equation. To solve the space homogeneous Boltzmann equation we use the fast algorithm presented in our previous paper [13]. A second order finite volume scheme is used for the transport equation.

The numerical example shows the effectiveness of our method. Moreover the algorithm can be very easily parallelized. The performance gain is nearly linear as the number of processors increases up to the size of the physical grid.

References

1. Aristov, V.V., Tcheremisin, F.G.: The conservative splitting method for the solution of a Boltzmann equation. Zh. Vychisl. Mat. i Mat. Fiz. **20**(1), 191–207, 262 (1980)

2. Beylkin, G.: On the fast fourier transform of functions with singularities. Appl. Comput. Harmon. Anal. **2**(4), 363–381 (1995)
3. Bird, G.A.: Molecular gas dynamics and the direct simulation of gas flows. Oxford Engineering Science Series, vol. 42. Clarendon Press, Oxford (1994)
4. Bird, G.A.: Recent advances and current challenges for DSMC. Comput. Math. Appl. **35**(1–2), 1–14 (1998)
5. Bobylev, A., Rjasanow, S.: Difference scheme for the Boltzmann equation based on the fast fourier transform. Eur. J. Mech. B Fluids **16**(2), 293–306 (1997)
6. Bobylev, A.V., Rjasanow, S.: Fast deterministic method of solving the Boltzmann equation for hard spheres. Eur. J. Mech. B Fluids **18**(5), 869–887 (1999)
7. Buet, C.: A discrete-velocity scheme for the Boltzmann operator of rarefied gas dynamics. Transp. Theory Statist. Phys. **25**(1), 33–60 (1996)
8. Cercignani, C.: The Boltzmann equation and its applications. Applied Mathematical Sciences, vol. 67. Springer, New York (1988)
9. Filbet, F., Mouhot, C., Pareschi, L.: Solving the Boltzmann equation in $N \log_2 N$. SIAM J. Sci. Comput. **28**(3), 1029–1053 (2006). (electronic), http://dx.doi.org/10.1137/050625175
10. Filbet, F., Russo, G.: High order numerical methods for the space non-homogeneous Boltzmann equation. J. Comput. Phys. **186**(2), 457–480 (2003). http://dx.doi.org/10.1016/S0021-9991(03)00065-2
11. Filbet, F., Russo, G.: Accurate numerical methods for the Boltzmann equation. In: Modeling and Computational Methods for Kinetic Equations. Model. Simul. Sci. Eng. Technol., pp. 117–145. Birkhäuser, Boston (2004)
12. Goldstein, D., Sturtevant, B., Broadwell, J.: Investigation of the motion of discrete-velocity gases. In: Muntz, E.P., et al. (eds.) Proceedings of the 16th International Symposium on RGD. Progress in Astronautics and Aeronautics, vol. 118, pp. 100–117 (1989)
13. Heintz, A., Kowalczyk, P., Grzhibovskis, R.: Fast numerical method for the Boltzmann equation on non-uniform grids. J. Comput. Phys. **227**(13), 6681–6695 (2008). http://dx.doi.org/10.1016/j.jcp.2008.03.028
14. Hirsch, C.: Numerical Computation of Internal and External Flows. Wiley, New York (1991)
15. Kosuge, S., Aoki, K., Takata, S.: Shock-wave structure for a binary gas mixture: finite-difference analysis of the Boltzmann equation for hard-sphere molecules. Eur. J. Mech. B Fluids **20**(1), 87–101 (2001)
16. Mouhot, C., Pareschi, L.: Fast algorithms for computing the Boltzmann collision operator. Math. Comp. **75**(256), 1833–1852 (2006). (electronic). http://dx.doi.org/10.1090/S0025-5718-06-01874-6
17. Nanbu, K.: Stochastic solution method of the master equation and the model Boltzmann equation. J. Phys. Soc. Japan **52**(8), 2654–2658 (1983)
18. Ohwada, T.: Structure of normal shock waves: direct numerical analysis of the Boltzmann equation for hard-sphere molecules. Phys. Fluids A **5**(1), 217–234 (1993)
19. Panferov, V.A., Heintz, A.G.: A new consistent discrete-velocity model for the Boltzmann equation. Math. Meth. Appl. Sci. **25**(7), 571–593 (2002)
20. Pareschi, L., Perthame, B.: A Fourier spectral method for homogeneous Boltzmann equations. In: Proceedings of the Second International Workshop on Nonlinear Kinetic Theories and Mathematical Aspects of Hyperbolic Systems (San Remo, 1994), vol. 25, pp. 369–382 (1996)

21. Pareschi, L., Russo, G.: Numerical solution of the Boltzmann equation. I. Spectrally accurate approximation of the collision operator. SIAM J. Numer. Anal. **37**(4), 1217–1245 (2000). (electronic)
22. Rjasanow, S., Wagner, W.: Stochastic numerics for the Boltzmann equation. Springer Series in Computational Mathematics, vol. 37. Springer, Berlin (2005)
23. Rogier, F., Schneider, J.: A direct method for solving the Boltzmann equation. Transp. Theory Statist. Phys. **23**(1–3), 313–338 (1994)

Simulating Phase Transition Dynamics on Non-trivial Domains

Łukasz Bolikowski and Maria Gokieli(✉)

Interdisciplinary Centre for Mathematical and Computational Modelling,
University of Warsaw, Prosta 69, 00-838 Warsaw, Poland
L.Bolikowski@icm.edu.pl, M.Gokieli@icm.edu.pl

Abstract. Our goal is to investigate the influence of the geometry and topology of the domain Ω on the solutions of the phase transition and other diffusion-driven phenomena in Ω, modeled e.g. by the Allen–Cahn, Cahn–Hilliard, reaction–diffusion equations. We present FEM numerical schemes for the Allen–Cahn and Cahn–Hilliard equation based on the Eyre's algorithm and present some numerical results on split and dumbbell domains.

Keywords: Diffusion · Cahn–Hilliard · Allen–Cahn · Stability · Finite element

1 Introduction and Motivation

The Allen–Cahn equation:

$$u_t - \varepsilon \Delta u = u - u^3 \quad \text{on } (0,\infty) \times \Omega \tag{1}$$

where $u = u(t,x)$, $u(0,\cdot) = u_0$ given in some domain Ω and with the homogenous Neumann boundary condition

$$\frac{\partial}{\partial n} u = 0 \quad \text{on } (0,\infty) \times \partial\Omega,$$

is of reaction-diffusion type. It is also a model of an order–disorder phase transition in alloys, u being then not a concentration, but a non-conserved order parameter [1]. In any interpretation the equation is a dissipative system, driven by the minimization of the free energy

$$\mathcal{F} = \int_\Omega \frac{\varepsilon}{2} |\nabla u(x)|^2 + \frac{1}{4}u^4 - \frac{1}{2}u^2 \, dx$$

in the Hilbert space $L^2(\Omega)$. It takes the so–called gradient form

$$\frac{du}{dt} = \frac{\delta \mathcal{F}}{\delta u}(u),$$

the Gateaux derivative here being taken in $L^2(\Omega)$.

A second classical model of phase transition phenomena, also closely related to diffusion, is the Cahn–Hilliard equation [2,3]. Roughly, it can be written as:

$$u_t - \Delta(-\varepsilon\Delta u - u + u^3) = 0 \quad \text{on } (0,\infty)\times\Omega \tag{2}$$

$u(0,\cdot) = u_0$ given in Ω and with the boundary condition

$$\frac{\partial}{\partial n}u = \frac{\partial}{\partial n}\Delta u = 0 \quad \text{on } (0,\infty)\times\partial\Omega.$$

Here, u is the concentration of one of the two components of an alloy or a mixture.

As shown by Fife [4], this equation is also a gradient system for the same free energy $\mathcal{F}$, but this times in the space $\tilde{H}^{-1}$, which the zero-average subspace of the dual of $H^1(\Omega)$.

The dynamics of both these equations are far from being trivial. One of their main characteristics is the coexistence very different time evolution scales: very fast and very slow. This is closely related to the attractor's structure of these systems, and in particular to their steady states. Let us give an example, for the simpler Allen–Cahn equation, considered in one space dimension only. Its steady states are then given by

$$-\varepsilon u_{xx} + u^3 - u = 0 \quad \text{on } (a,b), \quad u'(a) = u'(b) = 0.$$

Apart from constant solutions 0, -1 and 1, there are many states which are close to 1 or -1 by the ends on the interval, and have one or more 'transition layers' or 'interfaces' between these almost constant states. If we understand by stability of these states their being or not 'attractive' for evolving solutions in their neighborhood, one gets immediately that 1 and -1 are stable and 0 is unstable (just considering the linearization of the left-hand-side operator around these states and its eigenvalues). It is known also that more layers in the steady state means more instability. The numerical experiments performed by Fusco and Hale in 1989 [5] seemed to indicate additionally stability of the one-layer solution. They discovered, however, by a theoretical study, that its apparent immobility was only an extremely slow evolution toward a constant state — the final transition occurring, on the contrary, instantaneously. The solution got the name of 'dormant instability'.

Things get more complicated when we pass to higher space dimensions. The influence of the domain's shape on the dynamics, and particularly on the character of the steady solutions, is extensively studied since the fundamental results of Casten and Holland [6] and of Matano [7]. They found, independently, that in a class of regions including convex domains and annuli, the only stable solution of the reaction-diffusion type Neumann problems were constants. Other shapes have been considered from then on. The most studied one was the dumbbell, see Fig. 1 ([8–14]). Other shapes have been considered in [13–16]. In [17] we studied this question domains, with a non-lipschitz boundary (see Fig. 1), obtaining existence of non-constant stable equilibria.

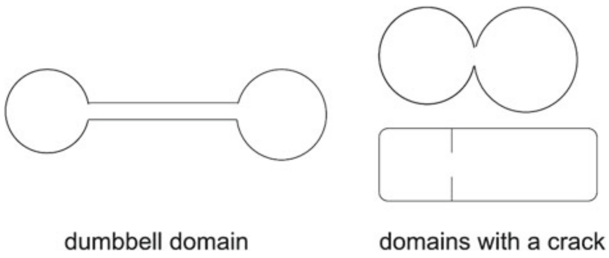


Fig. 1. Some shapes for which the question of reaction–diffusion/Allen–Cahn dynamics have been studied theoretically.

The same question appears for the Cahn–Hilliard equation, it has been studied e.g. in [18–22], although the domain shape influence is very little known for this case.

Let us also note that both equation coupled were also proposed as a model of two simultaneously occurring phase transitions in alloys [23].

Numerical simulations for phase transitions are still few, and inexistant, as far as we know, for complex domain shapes.

In [24] and [25], we proposed a numerical scheme and a solver for a system of both equations with a more sophisticated nonlinearity. Special cases included both equations taken separately. This fully implicit, nonlinear scheme was based on a splitting method of Lions and Mercier [26]. Previous numerical studies of phase transitions (see e.g. [27,28] and references therein for the method, [29,30] for its application) were based on the same idea.

A much simpler and faster method, even if based on a similar splitting, has been proposed by Eyre [31] and exploited by Vollmayr–Lee and Rutenberg in [32]. They show that with an appropriate choice of parameters, this scheme is semi-implicit (linearly implicit), and still stays a good approximation of the original equations. This method has been used to get a finite difference scheme on a square in [33,34]. Actually, for simple rectangular domains this semi implicit scheme can be solved via the discrete cosine transform. This would be fast and effective, but inappriopriate for the object of our interest, i.e. complex domain shapes. This is why we turn to the free element method.

We thus apply here the idea of Eyre, Vollmayr–Lee and Rutenberg to get a finite element scheme solving both equations (Sect. 2) and show some numerical results in split domains (Sect. 3). We hope that the conference will be an impulse to continue and broaden these first experiments. In a forthcoming [35] we will present a detailed proof of convergence and a solver for this method in a more general setting.

2 Discrete Scheme

We assume that Ω is polygonal. In Ω we introduce a family of quasi–uniform triangulations $T_h(\Omega)$ consisting of elements that are tetrahedrons in 3D case or triangles in 2D case, cf. [36] or Definition 4.4.13, p.106 in [37]. We call the parameter of triangulation, and denote by h, the maximum over the diameters of all the triangles.

We work with the simple generic case of linear finite elements over $T_h(\Omega)$, i.e. we introduce the space

$$V^h = \{v \in C(\overline{\Omega}) : \ v_{|T} \in P_1(T) \ \ \forall T \in T_h(\Omega)\},$$

where $P_1(T)$ is the set of linear polynomials over the element $T \in T_h(\Omega)$. The set of all nodal points, i.e. the vertices of elements of $T_h(\Omega)$, is denoted by Ω_h.

2.1 The Scheme for Allen–Cahn

We present the scheme for (1). Actually, the functions u^3 and u could be replaced by any increasing, locally lipschitz functions, but let them be fixed for clarity. We use the scheme proposed in [32] with $a_2 = 0$, $a_3 = 1$ and $a_1 = a$.

Let $\tilde{u}$ be the value from the previous time step. The unknown value u is obtained from:

$$\frac{u - \tilde{u}}{\tau} - \varepsilon \Delta u = (1 - a)u + a\tilde{u} - \tilde{u}^3. \tag{3}$$

Translated into the finite element method, (3) is: knowing $\tilde{v} = v^{n-1} \in V^h$, find $v = v^n \in V^h$ such that for all test functions $\phi \in V^h$

$$(1 + (a-1)\tau) \int_{\Omega_h} v\phi + \tau\varepsilon \int_{\Omega_h} \nabla v \nabla \phi = (1 + a\tau) \int_{\Omega_h} \tilde{v}\phi - \tau \int_{\Omega_h} \tilde{v}^3 \phi. \tag{4}$$

This leads to a linear system of equations with a positive definite matrix, having a unique solution for all n. As shown in [32], for $a \geq 4$ this scheme is gradient stable, i.e. it has the property of conserving the gradient structure of the problem:

$$\mathcal{F}(v^n) \leq \mathcal{F}(v^{n-1}).$$

Another property which is important from the modeling point of view is the maximum principle: the solution stays in the interval $[-1, 1]$.

The linear system if equations that we obtain from (4) can be solved by a direct or an iterative method. We actually use the direct Cholesky decomposition. This part can be parallelized.

2.2 The Scheme for Cahn–Hilliard

We re-write (2) as:

$$\begin{cases} u_t = \Delta w \\ w \ = -\varepsilon^2 \Delta u + u^3 - u \end{cases} \tag{5}$$

with the same boundary and initial conditions. The splitting allows to work with test functions taken only from $H^1(\Omega)$, and, in the discrete case, from V^h — this means that we can use only standard P_1 finite elements.

The weak formulation of the problem is then:

$$\begin{cases} \int_\Omega u_t\varphi + \int_\Omega \nabla w \nabla \varphi \quad = 0 \\ \int_\Omega w\psi - \varepsilon^2 \int_\Omega \nabla u \nabla \psi - \int_\Omega (u^3 - u)\psi = 0 \end{cases} \tag{6}$$

which should be satisfied for all $\phi, \psi \in L^\infty(0,T;\ H^1(\Omega))$. See e.g. [24] for the details of the weak formulation.

Time discretization is now done as previously, by substituting u with $a\tilde{u} + (1-a)u$, where $\tilde{u}$ is the value from the previous time step. This gives the following scheme: knowing $\tilde{u} = u^{n-1} \in V^h$, find $(u,w) = (u^n, w^n) \in V^h \times V^h$ such that for all pairs of test functions $(\phi,\psi) \in V^h \times V^h$

$$\begin{cases} \int_{\Omega_h} \frac{u-\tilde{u}}{\tau}\varphi + \int_{\Omega_h} \nabla w \nabla \varphi \quad = 0 \\ \int_{\Omega_h} w\psi - \varepsilon^2 \int_{\Omega_h} \nabla u \nabla \psi - \int_{\Omega_h} (\tilde{u}^3 - a\tilde{u} - (1-a)u)\psi = 0, \end{cases}$$

or

$$\begin{cases} \int_{\Omega_h} u\varphi + \tau \int_{\Omega_h} \nabla w \nabla \varphi \quad = \int_{\Omega_h} \tilde{u}\phi \\ \int_{\Omega_h} w\psi + (1-a)\int_{\Omega_h} u\phi - \varepsilon^2 \int_{\Omega_h} \nabla u \nabla \psi = \int_{\Omega_h} (\tilde{u}^3 - a\tilde{u})\psi = 0. \end{cases} \tag{7}$$

This is again a semi-implicit scheme, leading to a linear system of equations, of unknown $(u,w) \in \mathbb{R}^{2N}$, with a regular matrix:

$$A = \begin{bmatrix} M\ \tau G \\ (1-a)M - \varepsilon G\ M \end{bmatrix},$$

where M is the mass matrix and G is the stiffness matrix; $\det A = \det(M^2 + \tau(a-1)MG + \tau\varepsilon G^2) > 0$, see the forthcoming [35] for details.

3 Numerical Experiments

3.1 Results for Allen–Cahn

The computations for the Allen–Cahn equation has been performed with use of the FreeFEM++ software on a domain which approximates non-Lipschitz domains: we took two nearly circular subdomains and joined them by one node to which we add two more very small elements, as shown on Fig. 2. On the right-hand-side there, we show the smallest of all holes considered in our experiments. More experiments with finer meshes would still be needed. However, the results of simulations that we present below are in perfect accordance with the theory established in [17]; it is thus clear that no numerical artefacts are observed here.

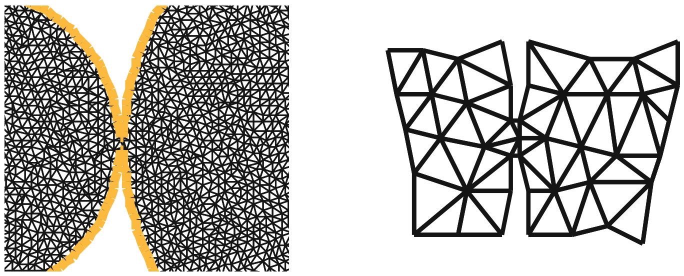


Fig. 2. Two close-up views on the FEM grid near the junction of the two subdomains.

Two experiments are shown in Fig. 3. They are performed with initial data taken as random perturbations of the constant 0, the constant unstable solution. The system evolves towards two different states: a constant solution (equal to -1) in case (a) and a nonconstant one, close to ± 1 in each subregion in case (b). None of this data seem to evolve. Whereas it is clear that the constant final state in (a) is stable, we continue the computations for (b) in order to check if any evolution towards a constant state can be seen. The non-constant state seems stable. This is actually true, as shown tn the theretical paper [17]. However, for a numerical confirmation and an insight into the dynamics of the studied process, we plot a measure of the rate of change of the function u in time, defined as:

$$m(t_n) = \int_{\Omega} \frac{|u^n - u^{n-1}|}{\tau} \tag{8}$$

Here n is the time step, u^n the numerical solution at time step n, i.e. at $t_n = n\tau$. The measure is more sensitive to changes of u than the rate of change of mass (where mass is defined as $\int_{\Omega} u$), in particular: $m(t_n) = 0$ implies $u^n \equiv u^{n-1}$ a.e. Figure 5 represents the measure for each of the experiments. Note that logarithmic scale is used for the vertical axes (Fig. 4).

One can see that the evolution speed is (i) nearly constant but *increasing* on big intervals of time, and (ii) changing rapidly on some very short intervals. This phenomenon is known to correspond to the evolution on the attractor, (i) following the invariant manifolds and (ii) near the unstable equilibria (see for instance [5] for the Allen–Cahn case). Our graphs show also a much slower and flatter part by the end of all the experiments, which makes one think there is a particular feature of the final state — and this is actually its stability.

3.2 Results for Cahn–Hilliard

For the Cahn–Hilliard equation (2), experiments were performed on a dumbbell domain. Here, the process is much slower and we focused up to now on the experiments visualizing the spinodal decomposition and the nucleation processes. The first one is occurring very rapidly, which again corresponds to the vicinity of the steady state in the attractor. Then, we observe a very slow process of

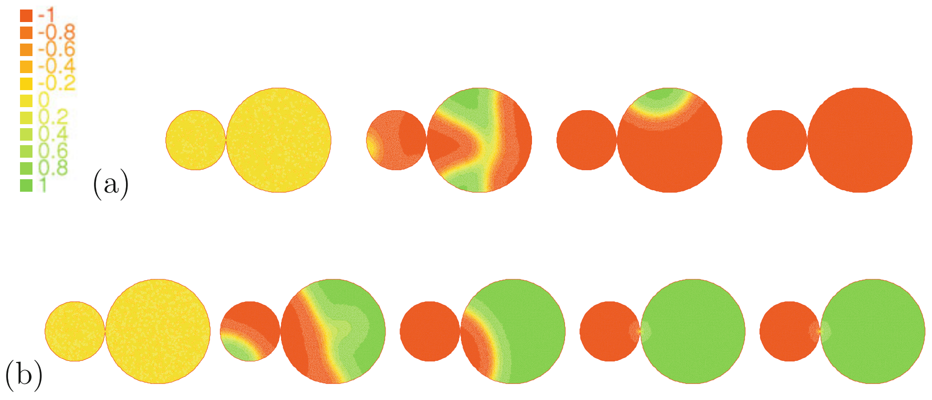


Fig. 3. Two experiments done with initial data $u_0 \approx 0$, with random perturbations. The evolution is visualized at $t = 0$, $t = 0$, $t = 10$, $t = 40$, $t = 100$ and $t = 1000$ in (a) and $t = 0$, $t = 10$, $t = 100$, $t = 200$, $t = 1000$ in (b). The numerical parameters are time–step $\tau = 0.001$, the diameter of the triangulation $h_{\max} = 0.00175$. The physical parameter $\varepsilon = 0.09$. The evolution is much slower.

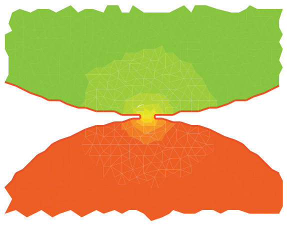


Fig. 4. The interface in the final stage from Fig. 3(b), a close-up.

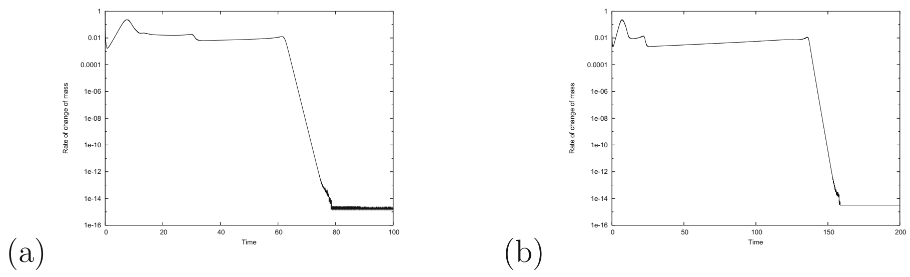


Fig. 5. The rate of mass change (8) for experiments of Fig. 3.

nucleation. It occurs at the same speed in the narrow channel and in the large subregions. The theoretical and numerical stability analysis is still to be done.

The experiment in Fig. 6 is performed with the numerical scheme (7) again using FreeFEM++.

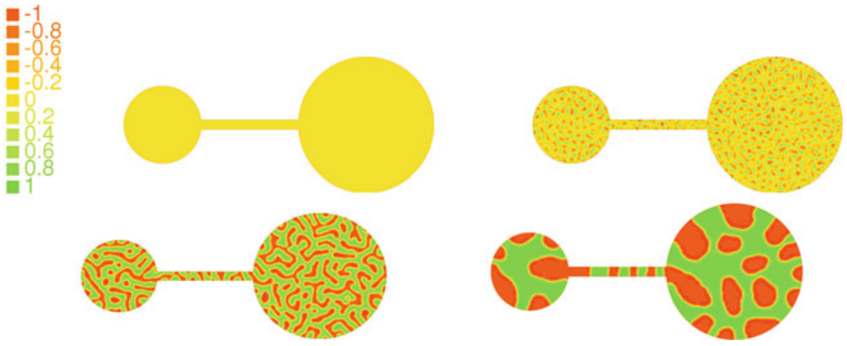


Fig. 6. The nucleation process for the Cahn–Hilliard equation. The geometry of the domain is similar to the previous one, but we add a channel between the two subregions. The evolution is visualized at $t = 0$, $t = 10$, $t = 700$, $t = 200$, $t = 5000$. The legend stands for both experiments. The parameters are as before time–step $\tau = 0.001$, the diameter of the triangulation $h_{\max} = 0.00175$, $\varepsilon = 0.09$.

References

1. Allen, S.M., Cahn, J.W.: A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening. Acta Metall. **27**(6), 1085–1095 (1979)
2. Cahn, J.W., Hilliard, J.E.: Free energy of a nonuniform system. i. interfacial free energy. J. Chem. Phys. **28**(2), 258–267 (1958)
3. Cahn, J.W., Hilliard, J.E.: Spinodal decomposition - a reprise. Acta Metall. **19**, 151–161 (1971)
4. Fife, P.C.: Models for phase separation and their mathematics (2000)
5. Fusco, G., Hale, J.K.: Slow-motion manifolds, dormant instability, and singular perturbations. J. Dynam. Differ. Eqn. **1**(1), 75–94 (1989)
6. Casten, R.G., Holland, C.J.: Instability results for reaction-diffusion equations with Neumann boundary conditions. J. Differ. Eqn. **27**, 266–273 (1978)
7. Matano, H.: Asymptotic behavior and stability of solutions of semilinear diffusion equations. Publ. Res. Inst. Math. Sci. **15**, 401–454 (1979)
8. Hale, J.K., Vegas, J.: A nonlinear parabolic equation with varying domain. Arch. Ration. Mech. Anal. **86**, 99–123 (1984)
9. Vegas, J.M.: Bifurcations caused by perturbing the domain in an elliptic equation. J. Differ. Eqn. **48**, 189–226 (1983)
10. Jimbo, S.: The singularly perturbed domain and the characterization for the eigenfunctions with Neumann boundary condition. J. Differ. Eqn. **77**(2), 322–350 (1989)
11. Jimbo, S.: Singular perturbation of domains and semilinear elliptic equations. III. Hokkaido Math. J. **33**(1), 11–45 (2004)
12. Jimbo, S., Morita, Y.: Remarks on the behavior of certain eigenvalues on a singularly perturbed domain with several thin channels. Commun. Partial Differ. Eqn. **17**(3/4), 523–552 (1992)
13. Arrieta, J.M., Carvalho, A.N., Lozada-Cruz, G.: Dynamics in Dumbbell domains. I: continuity of the set of equilibria. J. Differ. Eqn. **231**(2), 551–597 (2006)

14. de Oliveira, L.A., Pereira, A.L., Pereira, M.C.: Continuity of attractors for a reaction-diffusion problem with respect to variations of the domain (2005)
15. Dancer, E.: The effect of domain shape on the number of positive solutions of certain nonlinear equations. J. Differ. Eqn. **74**(1), 120–156 (1988)
16. Daners, D.: Dirichlet problems on varying domains. J. Differ. Eqn. **188**(2), 591–624 (2003)
17. Bolikowski, Ł., Gokieli, M., Varchon, N.: The Neumann problem in an irregular domain. Interfaces Free Bound. **12**(4), 443–462 (2010)
18. Alikakos, N.D., Fusco, G.: Slow dynamics for the Cahn-Hilliard equation in higher space dimensions: The motion of bubbles. Arch. Ration. Mech. Anal. **141**(1), 1–61 (1998)
19. Alikakos, N., Fusco, G., Karali, G.: Motion of bubbles towards the boundary for the Cahn-Hilliard equation. Eur. J. Appl. Math. **15**(1), 103–124 (2004)
20. Alikakos, N., Bates, P.W., Fusco, G.: Slow motion for the Cahn-Hilliard equation in one space dimension. J. Differ. Eqn. **90**(1), 81–135 (1991)
21. Bates, P.W., Fusco, G.: Equilibria with many nuclei for the Cahn-Hilliard equation. J. Differ. Eqn. **160**(2), 283–356 (2000)
22. Sander, E., Wanner, T.: Unexpectedly linear behavior for the Cahn-Hilliard equation. SIAM J. Appl. Math. **60**(6), 2182–2202 (2000). (electronic)
23. Cahn, J.W., Novick-Cohen, A.: Evolution equations for phase separation and ordering in binary alloys. J. Stat. Phys. **76**(3–4), 877–909 (1992)
24. Gokieli, M., Marcinkowski, L.: Discrete approximation of the Cahn-Hilliard/Allen-Cahn system with logarithmic entropy. Japan J. Ind. Appl. Math. **20**(3), 321–351 (2003)
25. Gokieli, M., Marcinkowski, L.: A solver for the finite element approximation scheme for the Cahn-Hilliard/Allen-Cahn system with logarithmic entropy. In: Aiki, T., et al. (eds.) Current Advances in Nonlinear Analysis and Related Topics. GAKUTO International Series. Mathematical Sciences and Applications 32. Gakkōtosho, Tokyo (2010)
26. Lions, P.L., Mercier, B.: Splitting algorithms for the sum of two nonlinear operators. SIAM J. Numer. Anal. **16**, 964–979 (1979)
27. Elliott, C.M., French, D.A.: A nonconforming finite element method for the two-dimensional Cahn-Hilliard equation. SIAM J. Numer. Anal. **26**(4), 884–903 (1989)
28. Blowey, J.F., Copetti, M.I.M., Elliott, C.M.: Numerical analysis of a model for phase separation of a multi-component alloy. IMA J. Numer. Anal. **16**(1), 233–280 (1996)
29. Kay, D., Welford, R.: A multigrid finite element solver for the Cahn-Hilliard equation. J. Comput. Phys. **212**(1), 288–304 (2006)
30. Wells, G.N., Kuhl, E., Garikipati, K.: A discontinuous galerkin method for the Cahn-Hilliard equation. J. Comput. Phys. **218**(2), 860–877 (2006)
31. Eyre, D.J.: An unconditionally stable one-step scheme for gradient systems. Preprint (1997), University of Utah, Salt Lake City
32. Vollmayr-Lee, B.P., Rutenberg, A.D.: Fast and accurate coarsening simulation with an unconditionally stable time step. Phys. Rev. E **68**(6), 066703 (2003)
33. De Mello, E., Teixeira da Silveira Filho, O.: Teixeira da Silveira Filho, O.: Numerical study of the Cahn-Hilliard equation in one, two and three dimensions. Physica A **347**, 429–443 (2005)
34. Cheng, M., Rutenberg, A.D.: Maximally fast coarsening algorithms. Phys. Rev. E **72**(5), 055701 (2005)

35. Bolikowski, Ł., Gokieli, M., Marcinkowski, L.: The Eyre's algorithm for solving phase transition models – analysis and application (in preparation)
36. Quarteroni, A., Valli, A.: Numerical Approximation of Partial Differential Equations. Springer, Berlin (1994)
37. Brenner, S.C., Scott, L.R.: The mathematical theory of finite element methods. Texts in Applied Mathematics, vol. 15. Springer, New York (1994)

Variable Block Multilevel Iterative Solution of General Sparse Linear Systems

Bruno Carpentieri[1(✉)], Jia Liao[1], and Masha Sosonkina[2]

[1] Institute of Mathematics and Computing Science, University of Groningen, 9747 AG Groningen, The Netherlands
{b.carpentieri,j.liao}@rug.nl

[2] Department of Modeling, Simulation and Visualization Engineering, Old Dominion University, Norfolk, VA 23529, USA
msosonki@odu.edu

Abstract. We present numerical results with a variable block multilevel incomplete LU factorization preconditioners for solving sparse linear systems arising, e.g., from the discretization of 2D and 3D partial differential equations on unstructured meshes. The proposed method automatically detects and exploits any available block structure in the matrix to maximize computational efficiency. Both sequential and parallel experiments are shown on selected matrix problems in different application areas, also against other standard preconditioners.

Keywords: Linear systems · Sparse matrices · Krylov methods · Algebraic preconditioners · Multilevel incomplete LU factorization · Graph compression

1 Introduction

We consider iterative solutions of sparse linear systems of the form

$$Ax = b \tag{1}$$

typically arising from the finite element, finite difference or finite volume discretization of partial differential equations (PDEs) on unstructured meshes. Here we denote by $A = [a_{ij}] \in \mathbb{R}^{n \times n}$ a large and sparse matrix, $b \in \mathbb{R}^n$ is a given right-hand side vector and $x \in \mathbb{R}^n$ is the unknown solution vector. We are particularly interested to design multilevel incomplete LU factorization methods for this problem class that can exploit any existing block structure in the matrix, and maximize computational efficiency.

It is known that many sparse matrices arising from the solution of systems of partial differential equations exhibit a natural block structure inherited from the underlying physical problem, e.g., when several physical quantities are associated with the same mesh node. If the, say ℓ, distinct variables associated with the same node are numbered consecutively, the permuted matrix has a sparse block structure with nonzero blocks of size $\ell \times \ell$. The blocks are usually fully dense, as variables at the same node are mutually coupled. We call this form of blocking a *perfect block ordering*. In multiphysics solution of coupled systems of PDEs, the blocks may correspond to sets of unknowns associated to different mathematical models, and thus may have variable size. Even in the case of general unstructured matrices, it is sometimes possible to compute *imperfect block orderings* by grouping together sets of columns and rows with a similar sparsity pattern, and treating some zero entries of the reordered matrix as nonzeros, and the nonzero blocks as dense, with a little sacrifice of memory [6].

In all these situations it is natural to consider block forms of multilevel factorization methods that exploit any available block structure (either perfect or imperfect) in the linear system, for the reasons explained below.

1. *Memory.* A clear advantage is to store the matrix as a collection of blocks using the variable block compressed sparse row (VBCSR) format, saving column indices and pointers for the block entries.
2. *Stability.* On indefinite problems, computing with blocks instead of single elements enables a better control of pivot breakdowns, near singularities, and other possible sources of numerical instabilities. Block ILU solvers may be used instead of pointwise ILU methods.
3. *Complexity.* Grouping variables in clusters, the Schur complement is smaller and hopefully the last reduced system is better conditioned and easier to solve.
4. *Efficiency.* A full block implementation, based on higher level optimized BLAS as computational kernels, may be designed leading to better flops to memory ratios on modern cache-based computer architectures.
5. *Cache effects.* Better cache reuse is possible for block algorithms.

2 The VBARMS Preconditioner

The Variable Block Algebraic Recursive Multilevel Solver (VBARMS) is a block variant of the ARMS preconditioner proposed by Saad and Suchomel in [8]. It can be used for preconditioning Krylov subspace solvers in the solution of sparse linear systems arising from the discretization of PDEs. An important feature of VBARMS is that it combines dense and sparse linear algebra kernels in the construction and in the application phase, achieving increased throughput during the computation and better reliability on realistic applications. The VBARMS method detects automatically any available block structure in the coefficient matrix A by using a graph compression algorithm, and maximally exploits this information during the factorization [1].

The complete pre-processing and factorization process of the VBARMS method can be summarized in the following steps.

Step 1. Compute a block ordering P_B of the coefficient matrix A, and permute A block-wise as

$$\widetilde{A} \approx P_B A P_B^T = \begin{bmatrix} \widetilde{A}_{11} & \widetilde{A}_{12} & \cdots & \widetilde{A}_{1p} \\ \widetilde{A}_{21} & \widetilde{A}_{22} & \cdots & \widetilde{A}_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \widetilde{A}_{p1} & \widetilde{A}_{p2} & \cdots & \widetilde{A}_{pp} \end{bmatrix}, \tag{2}$$

where the $n_i \times n_i$ diagonal blocks $\widetilde{A}_{ii}$, for $i = 1, \ldots, p$ are typically small and dense, and the off-diagonal blocks $\widetilde{A}_{ij}$ have size $n_i \times n_j$. We use a graph compression algorithm proposed by Saad, described in [6], for discovering any perfect or imperfect block structure in A, according to the definition introduced in Sect. 1.

Step 2. Scale the matrix $\widetilde{A}$ in the form $D_1 P_B A P_B^T D_2$, with two diagonal matrices D_1 and D_2, so that the 1-norm of the largest entry in each row and column is smaller or equal than one.

Step 3. Build the quotient graph $\mathcal{G}/\mathcal{B}$ of $\widetilde{A}$, where $\mathcal{B}$ denotes the partition into blocks (2). The quotient graph is obtained by coalescing the vertices assigned to block $\widetilde{A}_{ii}$ into a supervertex Y_i, for every $i = 1, \ldots, p$. Formally, we may define the quotient graph $\mathcal{G}/\mathcal{B}$ as the pair $\{V_{\mathcal{B}}, E_{\mathcal{B}}\}$ with

$$V_{\mathcal{B}} = \{Y_1, \ldots, Y_p\}, E_{\mathcal{B}} = \{(Y_i, Y_j) \,|\exists v \in Y_i, w \in Y_j \ s.t. (v, w) \in E\}.$$

An edge connects any supervertex Y_i to another supervertex Y_j iff there exists an edge from a vertex in A_{ii} to a vertex in A_{jj} in the graph $\{V, E\}$ of A.

Step 4. Find the independent-set ordering P_I from the quotient graph $\mathcal{G}/\mathcal{B} = \{V_{\mathcal{B}}, E_{\mathcal{B}}\}$, and apply the permutation to the matrix at Step 2 as

$$P_I D_1 P_B A P_B^T D_2 P_I^T = \begin{pmatrix} D & F \\ E & C \end{pmatrix}. \tag{3}$$

The upper left-most matrix D in the partitioning (3) is block diagonal,

$$D = \begin{pmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_\ell \end{pmatrix},$$

but due to the block permutation, the blocks D_i on the diagonal of D are block sparse matrices, as opposed to simply sparse matrices in other forms of multilevel ILU, e.g., in ARMS [8]. The matrices F, E, C are block sparse because of the same reasons.

We use a simple form of weighted greedy algorithm for computing the ordering P_I. The algorithm is the same as the one used in ARMS, and described in [8]. It consists of traversing the vertices $\mathcal{G}/\mathcal{B}$ in the natural order $1, 2, \ldots, n$, marking each visited vertex v and all of its nearest neighbors connected to v by an edge and adding v and each visited node that is not already marked to the independent set. We assign the weight $\|Y\|_F$ to each supervertex Y.

Step 5. Compute the block LU factorization of matrix (3)

$$\begin{pmatrix} D & F \\ E & C \end{pmatrix} = \begin{pmatrix} L & 0 \\ EU^{-1} & I \end{pmatrix} \times \begin{pmatrix} U & L^{-1}F \\ 0 & A_1 \end{pmatrix}, \tag{4}$$

where I is the identity matrix of appropriate size, L and U are the triangular factors of the (block) LU factorization of D, and A_1 is the Schur complement

$$A_1 = C - ED^{-1}F. \tag{5}$$

Also A_1 is block sparse; it has the same block partitioning of C.

Steps 3–5 can be repeated on the reduced system a few times until the Schur complement is small enough. Denote by A_ℓ the reduced Schur complement matrix at level ℓ, for $\ell > 1$. After scaling and preordering A_ℓ, a system with the matrix

$$P_I^{(\ell)} D_1^{(\ell)} A_\ell D_2^{(\ell)} (P_I^{(\ell)})^T = \begin{pmatrix} D_\ell & F_\ell \\ E_\ell & C_\ell \end{pmatrix} = \begin{pmatrix} L_\ell & 0 \\ E_\ell U_\ell^{-1} & I \end{pmatrix} \times \begin{pmatrix} U_\ell & L_\ell^{-1} F_\ell \\ 0 & A_{\ell+1} \end{pmatrix} \tag{6}$$

needs to be solved, with

$$A_{\ell+1} = C_\ell - E_\ell D_\ell^{-1} F_\ell. \tag{7}$$

Calling

$$x_\ell = \begin{pmatrix} y_\ell \\ z_\ell \end{pmatrix}, b_\ell = \begin{pmatrix} f_\ell \\ g_\ell \end{pmatrix}$$

the unknown solution vector and the right-hand side vector of system (6), the solution process with the above multilevel VBARMS factorization consists of level-by-level forward elimination followed by an exact solution on the last reduced system and suitable inverse permutation.

In VBARMS we perform the factorization approximately for memory efficiency. We use block ILU factorization with threshold to invert inexactly both the upper leftmost matrix $D_\ell \approx \bar{L}_\ell \bar{U}_\ell$ (with $\bar{L} \approx L$ and $\bar{U} \approx U$) at each level ℓ, and the last level Schur complement matrix $A_{\ell_{max}} \approx \bar{L}_S \bar{U}_S$. It is important to note that in this study the factorization is performed differently than in the original implementation of VBARMS proposed in [1], for better efficiency. We use a block variant of the ILUT algorithm based on the ikj-version of the Gaussian elimination process. The index k in the inner loop runs from 1 to $\min(m, i-1)$, where m is the size of the independent set, instead of from 1 to $i-1$ as is done in standard Gaussian elimination. This results in a decomposition $D_\ell \approx \bar{L}_\ell \bar{U}_\ell$, and an approximation to $\bar{L}_\ell^{-1} F_\ell$ is also computed. In a second loop, an approximation $E_\ell \bar{U}_\ell^{-1}$ and an approximation of the Schur complement matrix $A_{\ell+1}$ are derived.

The block ILU method used in VBARMS is a straightforward block variant of the one-level pointwise ILUT algorithm. The main difference is that we drop blocks $B \in \mathbb{R}^{m_B \times n_B}$ in $\bar{L}_\ell$, $\bar{U}_\ell$, $\bar{L}_S$, $\bar{U}_S$ whenever $\frac{\|B\|_F}{m_B \cdot n_B} < t$, for a given user-defined threshold t. The block pivots in block ILU are inverted exactly by using GE with partial pivoting. Finally, in assembling the Schur complement matrix $A_{\ell+1}$ at level ℓ, we take advantage of the finest block structure of D_ℓ, F_ℓ, E_ℓ, C_ℓ, imposed by the block ordering P_B on the small (usually dense) blocks in the diagonal blocks of D_ℓ and the corresponding small off-diagonal blocks in E_ℓ and F_ℓ; we call optimized level-3 BLAS routines [3] for computing $A_{\ell+1}$ in Eq. (7). Small blocks are dropped in the Schur complement at every level. The same threshold is applied in all these operations. The algorithm may breakdown due to singular blocks encountered in the block ILU factorization of the last level Schur complement $A_{\ell_{max}}$ and of the upper leftmost matrices D_ℓ (at each level ℓ). In our experiments we observed breakdowns only if the average block density of the block ordering computed at Step 1 is low, say below 60 %.

The implementation of the VBARMS method is developed in the C language and is based upon the ARMS code available in the ITSOL package [4]. The compressed sparse storage format of ARMS is adapted to store block vectors and matrices as a collection of contiguous nonzero dense blocks. In our implementation, the approximate transformation matrices $E_\ell \bar{U}_\ell^{-1}$ and $\bar{L}_\ell^{-1} F_l$ appearing in Eq. (6), at step ℓ, are temporarily stored in the VBCSR format, but they are discarded from the memory after assembling $A_{\ell+1}$. We only store the factors $\bar{L}_\ell, \bar{U}_\ell$ at each reduction ℓ, and $\bar{L}_S, \bar{U}_S$, because these are the matrices needed in the solving phase. Finally, we explicitly permute the matrix after Step 1 at the first level as well as the matrices involved in the factorization at each new reordering step.

3 Parallel Implementation

The parallelization of the VBARMS method is carried out by partitioning the quotient graph $\mathcal{G}/\mathcal{B}$ into separate subdomains, each assigned to a different processing unit. Each node of the quotient graph, assigned to a subdomain i, represents one physical node of the underlying mesh and a cluster of fully coupled unknowns of the linear system.

Following the parallel framework described in [5], local nodes are distinguished into *interior nodes*, those coupled only with local variables by the equations, and *interface nodes*, those that may be coupled with local variables stored on processor i and with remote variables stored on other processors. Accordingly, the vector of local unknowns x_i is split in two parts: the subvector u_i of internal nodes followed by the subvector y_i of local interface variables. The right-hand side b_i is split conformably in two subvectors f_i and g_i,

$$x_i = \begin{pmatrix} u_i \\ y_i \end{pmatrix}, \quad b_i = \begin{pmatrix} f_i \\ g_i \end{pmatrix}.$$

The rows of the linear system assigned by the graph partitioning to the subdomain i are also split in two separate contributions, a local matrix A_i which acts on the local variables $x_i = (u_i, y_i)^T$, and an interface matrix U_i which acts on the remote subvectors of external interface variables, denoted as $y_{i,ext}$. Therefore, the local equations on processor i may be written as

$$A_i x_i + U_{i,ext} y_{i,ext} = b_i$$

or also

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}, \tag{8}$$

where N_i is the set of subdomains that are neighbors to subdomain i and the submatrix $E_{ij} y_j$ accounts for the contribution to the local equation from the jth neighboring subdomain.

In our experiments, we use the VBARMS method to invert approximately the matrix A_i on each processor. The solves with the matrices A_i are all performed indepedently on all processors, clearly achieving high degree of parallelism. We also consider a parallel implementation of VBARMS based on Schur complement approaches. In Eq. (8), we can eliminate the vector of interior unknowns u_i from the first equations to obtain the local Schur complement system:

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i \equiv g_i',$$

where S_i is the local Schur complement matrix

$$S_i = C_i - E_i B_i^{-1} F_i.$$

Writing all the local Schur complement equations together results in the global Schur complement system:

$$\begin{pmatrix} S_1 & E_{12} & \dots & E_{1p} \\ E_{21} & S_2 & \dots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p-1,2} & \dots & S_p \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = \begin{pmatrix} g_1' \\ g_2' \\ \vdots \\ g_p' \end{pmatrix}, \tag{9}$$

where the off-diagonal matrices E_{ij} are already available from the data structure of the distributed sparse linear system. One preconditioning step consists in solving the global system (9) approximately, and computing the u_i variables from the local equations as

$$u_i = B_i^{-1}[f_i - F_i y_i].$$

This operation requires only a local solve. In our study we solve the global system (9) approximately by a few steps of GMRES preconditioned by block Jacobi, where the diagonal blocks are the local Schur complements S_i. The factorization

$$S_i = L_{S_i} U_{S_i}$$

is obtained from the LU factorization of the local matrix A_i,

$$A_i = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix}.$$

This operations does not require any additional computation, as the factorization of A_i is required for computing the u_i variables.

In the next section, we report on numerical experiments to illustrate the performance of the VBARMS code for solving realistic applications.

4 Numerical Results

We applied the VBARMS method on a set of sparse linear systems $Ax = b$ arising from different application areas. For each linear system, we give in Table 1 the size, application field, number of nonzero entries and the characteristics of the block ordering computed by the compression algorithm. The column *b-size* shows the average block size of A after the compression, and the column *b-density* shows the ratio of the number of nonzero entries in A before and after the compression. It is *b-density* $=$ *100*% if the graph compression algorithm finds a perfect block structure in A with fully dense nonzero blocks, whereas *b-density* $<$ *100*% means that some zero entries in the blocks are treated as nonzeros in VBARMS. The matrix problems are extracted from Tim Davis' matrix collection at the University of Florida [2], except for the problems denoted `RAE, STA004, STA008` that were kindly supplied by Professor Aldo Bonfiglioli at University of Basilicata, Italy.

We use an algorithm proposed by Saad in [6] that discovers perfect or imperfect block structures in the system by comparing the sparsity patterns of consecutive rows. Briefly, the algorithm computes the inner product of a reference row i with row j, for $j > i$; this product gives the cosine between the two rows. If the corresponding angle is smaller than a given threshold τ, row j will be assigned to the group of rows with similar pattern to row i. This is repeated for every row $i = 1, ..., n$ and $j > i$, but only for rows i and j that are not assigned to any group yet, otherwise the inner product is skipped. In our experiments, we initially set $\tau = 1$ to find sets of rows and columns having the same pattern and discover the presence of fully dense blocks in the matrix. We tested different values for τ, ranging from 0.7 to 1 on these two problems; with very little sacrifice in memory, it was possible to obtain larger blocks with still high density around 90 %. The computed block ordering was used in our experiments with the VBARMS.

We compared the sequential and parallel VBARMS preconditioners with the original ARMS code [8] and the standard ILUT methods [7]. For the sequential version of ARMS and ILUT, we used the implementations available in the ITSOL package [4]; for the parallel version we used the pARMS package [4]. Prior to the iterative solution, we scaled the system by rows and columns so that the modulus of each entry of the scaled coefficient matrix was smaller than one. By an abuse of notation we continue denoting by A the compressed matrix in the experiments with VBARMS. We used physical right-hand sides

Table 1. Set and characteristics of test matrix problems.

Name	Size	Application	nnz(A)	b-size	b-density (%)
RAE	52995	Turbulence analysis	1748266	4.00	97
CT20STIF	52329	Engine block	2600295	2.61	100
RAEFSKY3	21200	Fluid structure interaction	1488768	8.00	100
VENKAT01	62424	2D Euler solver	1717792	4.00	100
BMW7ST	141347	Car body	7318399	4.63	100
AUDIKW_1	943695	Structural problem	77651847	3.00	100
LDOOR	952203	Structural problem	42493817	6.92	100
STA004	891815	Fluid dynamics	55902869	1.56	100
STA008	891815	Fluid dynamics	55902989	1.56	100

Table 2. Performance comparison of sequential VBARMS versus ARMS.

Matrix	Compression	Method	P-T	I-T	Total	Its	M-cost
RAE	$\tau = 0.80$,	VBARMS	4.51	0.62	5.13	15	4.62
	b-density = 95.83 %,	ARMS	68.95	73.36	142.30	+1000	29.26
	b-size = 4.67.	ILUT	132.06	106.12	238.18	+1000	49.99
CT20STIF	$\tau = 0.80$	VBARMS	1.51	1.68	3.19	39	2.42
	b-density = 86.76 %	ARMS	18.63	40.81	59.44	+1000	8.27
	b-size = 5.01.	ILUT	90.60	49.13	139.73	+1000	11.86
RAEFSKY3	$\tau = 0.80$,	VBARMS	0.77	0.04	0.81	3	2.00
	b-density = 95.22 %	ARMS	5.07	0.05	5.12	3	4.01
	b-size = 8.63.	ILUT	1.81	0.06	1.87	6	2.39
VENKAT01	$\tau = 0.80$,	VBARMS	1.74	0.21	1.96	5	2.56
	b-density = 100.00 %	ARMS	0.72	0.16	0.88	6	2.32
	b-size = 4.00.	ILUT	1.81	0.09	1.27	4	4.18
BMW7ST	$\tau = 0.80$,	VBARMS	6.54	0.23	6.77	2	3.67
	b-density = 95.26 %	ARMS	22.65	73.44	96.10	+1000	3.73
	b-size = 5.90.	ILUT	48.13	103.97	152.10	+1000	8.37

b when these were available, otherwise we set $b = Ae$ with $e = [1, \ldots, 1]^T$. For every run, we recorded the solution time from the start of the solve until either the initial residual was reduced by six orders of magnitude or no convergence was achieved after the prescribed maximum number of iterations of the flexible GMRES (FGMRES) method [7]. We restarted FGMRES every 20 inner iterations on the small problems (Tables 2–3), and every 100 inner iterations on the larger problems (Table 4). One important parameter to tune in VBARMS is the dropping threshold t. Small blocks $B \in \mathbb{R}^{m_B \times n_B}$ are dropped in the incomplete factors $\bar{L}_\ell$, $\bar{U}_\ell$, $\bar{L}_S$, $\bar{U}_S$ and in the last level Scur complement matrix $A_{\ell_{max}}$ whenever $\frac{\|B\|_F}{m_B \cdot n_B} < t$. For each matrix problem, we tested different values for the dropping parameter t in VBARMS, starting from $t = 0.1$ and decrementing it by a factor of 10 in each run; we selected the value of t which gave the best convergence result for the given problem. Finally, the number of levels of

Table 3. Performance analysis ofparallel VBARMS on 8 processors.

Matrix	Compression	Method	P-T	I-T	Total	Its	M-cost
RAE	$\tau = 0.80$,	SCHUR+VBARMS	1.56	98.00	99.56	714	7.19
	b-density = 95.83 %,	BJ+VBARMS	1.25	3.79	5.03	251	3.46
	b-size = 4.67 %.	BJ+ARMS	2.43	22.14	24.57	+1000	9.81
		BJ+ILUT	3.30	27.85	31.15	+1000	13.36
CT20STIF	$\tau = 0.80$,	SCHUR+VBARMS	0.38	1.82	2.20	40	2.59
	b-density = 86.76 %,	BJ+VBARMS	0.20	0.62	0.82	37	1.96
	b-size = 5.01.	BJ+ARMS	0.80	28.59	29.39	+1000	6.93
		BJ+ILUT	0.54	18.73	19.26	+1000	3.70
RAEFSKYY3	$\tau = 0.80$,	SCHUR+VBARMS	0.21	0.03	0.23	4	1.85
	b-density = 95.22 %,	BJ+VBARMS	0.09	0.04	0.13	3	1.70
	b-size = 8.63.	BJ+ARMS	0.13	0.24	0.37	6	2.45
		BJ+ILUT	0.08	0.32	0.40	8	1.80
VENKAT01	$\tau = 0.80$,	SCHUR+VBARMS	0.40	2.60	3.01	131	2.64
	b-density = 100.00 %,	BJ+VBARMS	0.29	0.30	0.59	13	2.43
	b-size = 4.00.	BJ+ARMS	0.45	2.70	3.15	14	10.78
		BJ+ILUT	0.20	0.24	0.44	13	3.96
BMW7ST	$\tau = 0.80$,	SCHUR+VBARMS	12.18	14.09	26.27	58	4.00
	b-density = 95.26 %,	BJ+VBARMS	0.09	0.51	1.41	5	2.63
	b-size = 5.90.	BJ+ARMS	3.95	57.47	61.41	+1000	4.34
		BJ+ILUT	24.12	88.36	112.48	+1000	9.75

Table 4. Performance comparison of BJ + VBARMS and ARMS on larger matices.

Matrix	Compression	Method	P-T	I-T	Total	Its	M-cost
AUDIKW_1	$\tau = 0.80$,	BJ+VBARMS	84.23	308.18	392.42	331	3.46
	b-density = 96.40 %,	BJ+ARMS	114.43	1785.02	1899.45	+3000	5.24
	b-size = 3.16.						
LDOOR	$\tau = 0.80$,	BJ+VBARMS	18.43	99.12	117.55	340	3.90
	b-density = 99.96 %,	BJ+ARMS	48.59	1194.43	1243.01	+3000	7.66
	b-size = 7.00.						
STA004	$\tau = 0.60$,	BJ+VBARMS	19.14	81.14	100.27	92	3.88
	b-density = 84.74 %,	BJ+ARMS	9.36	65.92	75.28	145	2.87
	b-size = 3.92.						
STA008	$\tau = 0.60$,	BJ+VBARMS	44.82	195.89	240.71	256	5.27
	b-density = 84.74 %,	BJ+ARMS	151.64	7740.94	7892.57	+3000	11.83
	b-size = 3.92.						

recursive factorization in VBARMS and ARMS were calculated automatically by the two codes, which stop when the Schur complement becomes too small to continue reducing the matrix. The maximum allowed size for the last level Schur complement matrix was set to 300. This value also determines the minimum size of the independent sets in the greedy algorithm.

For each experiment, we report the time cost for computing the factorization (column "P-T") and for solving the linear system (column "I-T"), the ratio of the total number of nonzeros in the factors to the number of nonzeros in the coefficient matrix A (column "M-cost"), and the number of FGMRES iterations (column "Its"). In Table 2 we report comparative results from our sequential runs. The results highlight the robustness of the VBARMS preconditioner. This is probably due to the better control of near-singularities of block ILU solvers, and to the better conditioning of the Schur complement matrices that are smaller and easier to invert. In our experiments on the small problems, we observed that the triangular factors computed by VBARMS were well conditioned; consequently, the triangular solves were numerically stable.

In Table 3 we show the parallel performance of VBARMS, also against parallel ARMS and ILUT on the same problems. In these experiments, we compare the block Jacobi preconditioner (denoted as BJ) with VBARMS, ARMS and ILUT as local solvers, and the Schur complement method (denoted as SCHUR). In the latter method, we use VBARMS as local solver and a fews steps of inner GMRES iterations for solving the global Schur complement system; precisely, the inner iterations are stopped after 100 steps or when the norm of the relative residual is decreased by two orders of magnitude. Note that the outer FGMRES iterations are stopped using the condition of the reduction of the initial residual by six orders of magnitude, consistently with the sequential runs. We refer the reader to Sect. 3 for a description of these preconditioners. We see that VBARMS can be more efficient and numerically more stable than ARMS. The results reported in Table 4 on larger problems confirm the trend. In our experiments, block Jacobi was surprisingly more robust than the Schur complement-based preconditioner. We found that the performance of the latter solver were strongly dependent on the strategy used for scaling the Schur complement matrix. In the experiments reported in Table 4 we scale the global Schur complement system prior to solving it. But this turns out not to be the best strategy for every problem, e.g., on the `VENKAT01` problem SCHUR converges in only 5 iterations without scaling, at approximately the same memory cost as BJ. We shall look more closely into the problem of finding an optimal parameter setting for the SCHUR solver in a separate study.

5 Concluding Remarks

We have shown that exposing dense matrix blocks during the factorization may lead to efficient and stable multilevel preconditioners. We are now developing parallel preconditioners based on the restricted additive Schwarz and multilevel Schur complement methods for solving large turbulence problems expressed in implicit Newton-Krylov formulation, that was the starting point of this study.

References

1. Carpentieri, B., Liao, J., Sosonkina, M.: VBARMS: a variable block variant of the algebraic multilevel solver for general linear systems. J. Comput. Appl. Math. (2013). ISSN **0377–0427**. doi:10.1016/j.cam.2013.04.036. 25 April 2013

2. Davis, T.: Sparse matrix collection (1994). http://www.cise.ufl.edu/research/sparse/matrices
3. Dongarra, J.J., Du Croz, J., Duff, I.S., Hammarling, S.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Softw. **16**, 1–17 (1990)
4. Li, N., Suchomel, B., Osei-Kuffuor, D., Saad, Y.: ITSOL: iterative solvers package
5. Li, Z., Saad, Y., Sosonkina, M.: pARMS: a parallel version of the algebraic recursive multilevel solver. Numer. Linear Algebra Appl. **10**, 485–509 (2003)
6. Saad, Y.: Finding exact and approximate block structures for ILU preconditioning. SIAM J. Sci. Comput. **24**(4), 1107–1123 (2002)
7. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. SIAM, Philadelphia (2003)
8. Saad, Y., Suchomel, B.: ARMS: an algebraic recursive multilevel solver for general sparse linear systems. Numer. Linear Algebra Appl. **9**(5), 359–378 (2002)

An Automatic Way of Finding Robust Elimination Trees for a Multi-frontal Sparse Solver for Radical 2D Hierarchical Meshes

Hassan AbouEisha[1], Piotr Gurgul[2], Anna Paszyńska[3], Maciek Paszyński[2](✉), Krzysztof Kuźnik[2], and Mikhail Moshkov[1]

[1] King Abdullah University of Science and Technology, Thuwal, Saudi Arabia
[2] AGH University of Science and Technology, Krakow, Poland
[3] Jagiellonian University, Krakow, Poland
paszynsk@agh.edu.pl
http://home.agh.edu.pl/~paszynsk

Abstract. In this paper we present a dynamic programming algorithm for finding optimal elimination trees for the multi-frontal direct solver algorithm executed over two dimensional meshes with point singularities. The elimination tree found by the optimization algorithm results in a linear computational cost of sequential direct solver. Based on the optimal elimination tree found by the optimization algorithm we construct heuristic sequential multi-frontal direct solver algorithm resulting in a linear computational cost as well as heuristic parallel multi-frontal direct solver algorithm resulting in a logarithmic computational cost. The resulting parallel algorithm is implemented on NVIDIA CUDA GPU architecture based on our graph-grammar approach.

Keywords: Parallel multi-frontal direct solver · Elimination tree · Dynamic programming · Adaptive finite element method · Graph grammar

1 Introduction

In this paper we present a dynamic programming algorithm for finding optimal elimination trees of multi-frontal direct solver algorithm executed over the computational grids resulting from adaptive finite element method simulations. In particular we focus on the two dimensional h refined grids generated with hierarchical shape functions [5]. The multi-frontal solver is the state of the art algorithm for solution of sparse matrices resulting from finite element method discretizations [1–3,6–9]. The multi-frontal solver algorithm browses the constructed elimination tree from leaves up to the root, merges frontal matrices generated at leaves nodes, and eliminates fully assembled degrees of freedom. The performance of the multi-frontal solver algorithm depends on the quality of the elimination tree. The computational cost of the multi-frontal solver algorithm for two dimensional regular grids is of the order of $O(N^{1.5})$ for the sequential version [4] and $O(NlogN)$ for the shared memory version [10], where N is the number of elements. Different elimination trees result in a different computational cost of the multi-frontal solver algorithm. The paper presents one dynamic programming algorithm that finds the optimal elimination tree for a given mesh with point singularities. The algorithm browses all the possible elimination trees and estimates the resulting cost of the multi-frontal algorithm for each of the tree. It utilized the dynamic programming concept to minimize its cost. The algorithm finds the optimal tree resulting in a linear cost of the multi-frontal solver algorithm. Based on the exhaustive search we propose two heuristic algorithms, sequential and parallel, that reconstructs the optimal tree in linear or logarithmic time, respectively. The parallel algorithm has been implemented on NVIDIA CUDA GPU with the graph grammar approach [10,12,13] The elimination trees found by our dynamic programming algorithm are the optimal one, since we performed the exhaustive search over all the possible elimination trees. We compare our heuristic algorithms constructed based on the dynamic programming search, with trees provided by the METIS library [11] from MUMPS solver [1–3].

2 Optimisation Algorithm

The search for the optimal elimination tree can be represented by the directed acyclic graph (DAG) presented in Fig. 1. The root node of the DAG represents the entire computational mesh, while child nodes represent possible partitions of the mesh. Each sub-domain is consider for the recursive partitions, and we get the tree of all possible partitions of the computational mesh. It is clear that some sub-branches of the DAG are identical, and some of them are denoted on Fig. 1 by red or green color. These sub-branches do not need to be regenerated, since we can use pointers to already generated identical sub-branch. The optimal

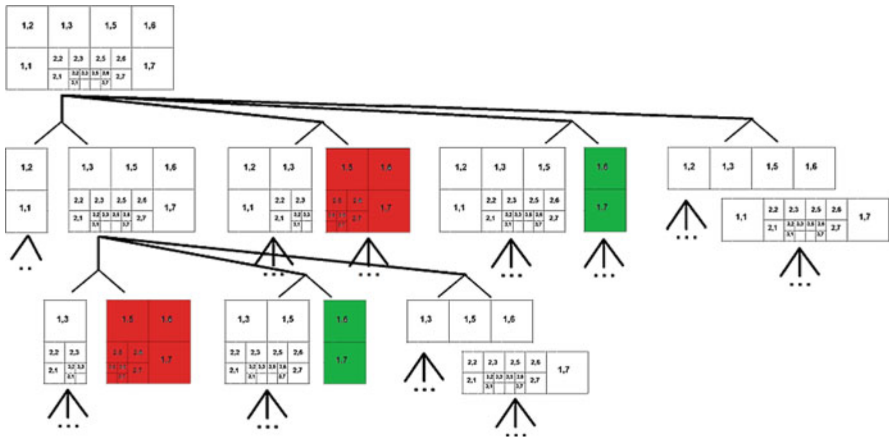


Fig. 1. The DAG representing all possible partitions of the mesh. (Color figure online)

elimination trees are represented as binary sub-trees of this DAG. The optimisation procedure utilize the cost function defined in the recursive way. The cost of processing the internal node is defined as

$$\begin{aligned}\text{Cost of processing internal node} = {} & \text{Cost of processing first child} + \\ & \text{Cost of processing second child} + \\ & \text{Cost of elimination of common interface} \end{aligned} \tag{1}$$

Each node of the elimination tree contains a frontal matrix with size b having some number a of fully assembled degrees of freedom. Leaf nodes contain element frontal matrices with fully assembled internal nodes which can be eliminated. The cost $C(a, b)$ of elimination of a fully assembled nodes from frontal matrix of size b is equal to

$$C(a, b) = \sum_{m=(b-a)+1}^{b} 3m^2 = \frac{a(6b^2 - 6ab + 6b + 2a^2 - 3a + 1)}{2} \tag{2}$$

This is just a number of operations for partial forward elimination algorithm. Given a geometric description of the finite element mesh, the dynamic programming algorithm works in two steps. In the first step, the DAG representing the subproblems and dependency relations between them is constructed. Afterwards, the algorithm continues its work by optimizing the DAG in a bottom-up approach. The construction of the DAG is performed as follows. The first step starts by adding a first node to the DAG corresponding to the initial mesh and the main problem. At any subsequent step $t > 1$, any unprocessed node is processed and this node is marked as processed. The algorithm terminates once all nodes are processed. The processing of a node is done by examining all partitions that are based on a straight lines (we call them later splitters) extending through the submesh corresponding to the node under consideration. For each such splitter, a pair of edges is drawn from the current node to the nodes corresponding to submeshes below (left of) and above (right of) this horizontal (vertical) splitter. After such processing, the current node is marked as processed. After building the DAG, the algorithm moves to the optimization stage based on a given cost function. This cost function specifies the cost of nodes in the DAG with zero outdegree (we call them sinks) in addition to determining the cost of other non-sink nodes by selecting best cost of all possible partitions. The algorithm begins by assigning cost to the sinks in the DAG. Then for all non-sink nodes whose all descendants are processed, each partition is assigned a cost function based on the splitter used in the partition and the resulting submeshes. The partitions with the optimal cost among all partitions are kept and other nodes are removed. The optimization procedure continues this way until reaching the root of the tree. The computational cost of the optimisation algorithm is $O(N^3)$.

In this section we present results of the optimisation procedure executed for the radical mesh [14], namely the rectangular mesh with one point sigularity located at the center of the bottom border. In the numerical experiments we focused on the heat transfer problem, however the direct solver algorithm is actually independent on the equation being solve. We utilized hierarchical shape functions from *hp* finite elements [5] and performed experiments for second, and fourth and order polynomials. The shape of the optimal elimination tree found by the optimization algorithm is presented in Fig. 2. This optimal elimination tree browses the radical mesh level by level, starting from the layer of elements surrounding the point singularity, up to the layer of boundary elements.

3 Heuristic Algorithm

Based on the results of the optimization procedure we propose the following heuristic algorithm. We focus here on the exemplary radical mesh refined towards point singularity, described in Figs. 3 and 4. The direct solver algorithm browses the elements of the h-refined computational mesh level by level, from the lowest level up to the top level. The algorithm utilizes a single frontal matrix, which size is proportional to the number of browsed nodes. Each row of the frontal matrix is associated with one mesh node. The solver algorithm finds the fully assembled nodes and places them at the top of the frontal matrix. Then, these rows are eliminated by subtracting from all the remaining rows. Intuitively, we can say that element interior nodes are fully assembled once the solver touches the element, an element edge nodes are fully assembled once the solver algorithm

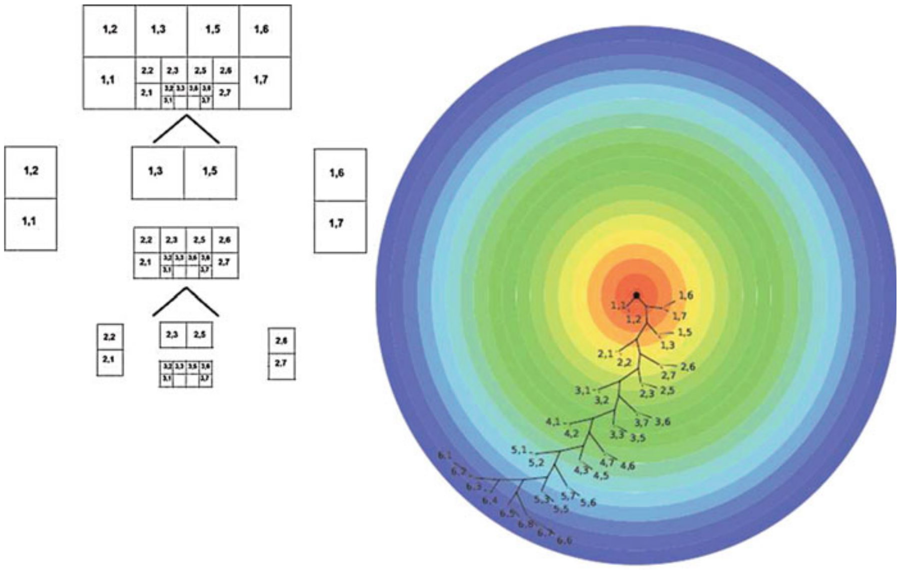


Fig. 2. The optimal elimination tree found by the optimization algorithm.

touches all elements adjacent to an edge, and finally element vertex nodes are fully assembled once the solver algorithm touches all elements having the vertex. Thus, on the bottom level we can eliminate nodes related to two interiors of the bottom elements, as well as the nodes located on the common edge and common bottom vertex, see left panel in Fig. 3. The remaining nodes must wait until the solver algorithm browses the elements from the next level, compare right panel in Fig. 3. The size of the frontal matrix is limited by the number of elements on a particular level, and this implies the linear computational cost of the solver algorithm. The computational cost of the heuristic algorithm as well as the direct solver algorithm is linear $O(N)$. We can also construct a heuristic algorithm for multi-frontal direct solver, as illustrated in Fig. 4. In the multi-frontal approach we construct an elimination tree with leaves related to the levels of the h refined mesh. After the elimination of nodes located inside the levels, we end up with the nodes located on the interfaces between levels. These interface nodes are process within the top binary elimination tree. The elimination tree can be process in sequential mode, where the linear computational cost of processing levels of the mesh is followed by the $rlogr$ cost for processing the interface nodes, where r is the number of levels in the mesh.

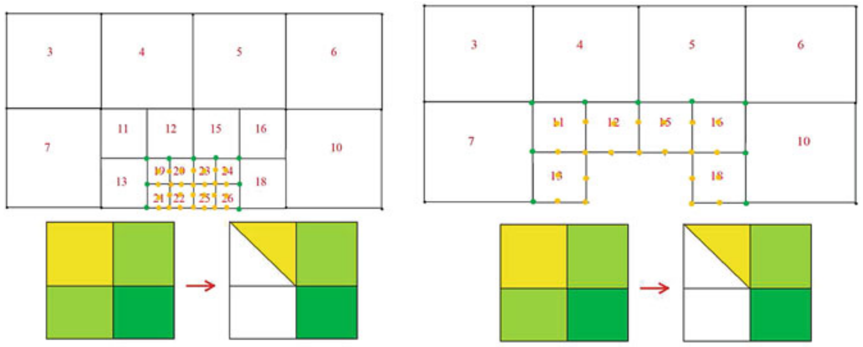


Fig. 3. Left panel: two finite element computational mesh with one singularity, and the first step of the solver algorithm. Right panel: the second step of the solver algorithm.

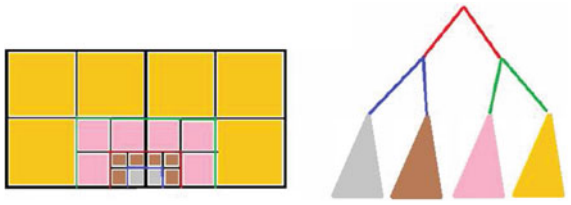


Fig. 4. Multi-frontal elimination pattern.

4 Numerical Results for Sequential Solver

In this section we compare the heuristic frontal algorithm constructed based on our optimization algorithm with multi-frontal MUMPS solver [1–3] with METIS library [11]. The numerical results for radical mesh for polynomial orders of approximation $p = 2$ and $p = 4$ are summarized in Fig. 5. They confirm the linear computational cost of our heuristic sequential algorithm. They also prove that the elimination tree for multi-frontal solver generated by METIS library is actually equivalent to the optimal elimination tree found by our heuristic algorithm. The number of degrees of freedom is small, however this strategy can be generalized for more singularities. In particular, we can perform this level by level elimination from bottom of the refinement tree up to the top, for each of the singularities. The total time will be equal to a sum of times for local singularities. To illustrate that process we also present the numerical results for the solver executed for two singularities, see Fig. 6. Our heuristic frontal algorithm for two singularities also delivers a linear cost, as well as it outperforms MUMPS with METIS for higher orders of approximations.

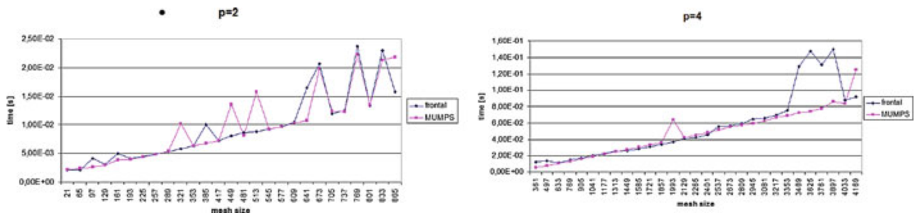


Fig. 5. Comparison of our sequential frontal heristic algorithm with MUMPS with METIS library for radical mesh for polynomial orders of approximation $p = 2, 4$.

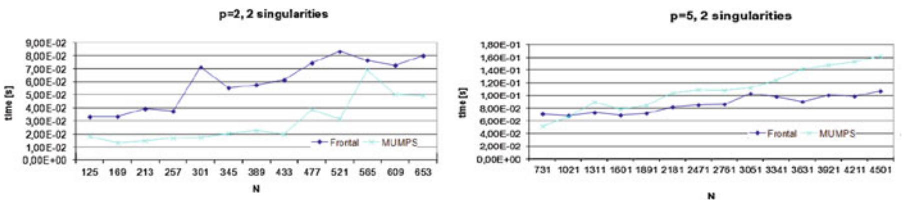


Fig. 6. Comparison of our sequential heristic algorithm with MUMPS with METIS library for radical mesh for polynomial orders of approximation $p = 2, 5$.

5 Graph Grammar Based Algorithm for Shared Memory Parallel Machine

In this section we present a heuristic parallel algorithm expressed by graph grammar model. The parallel algorithm has been constructed by analysing the

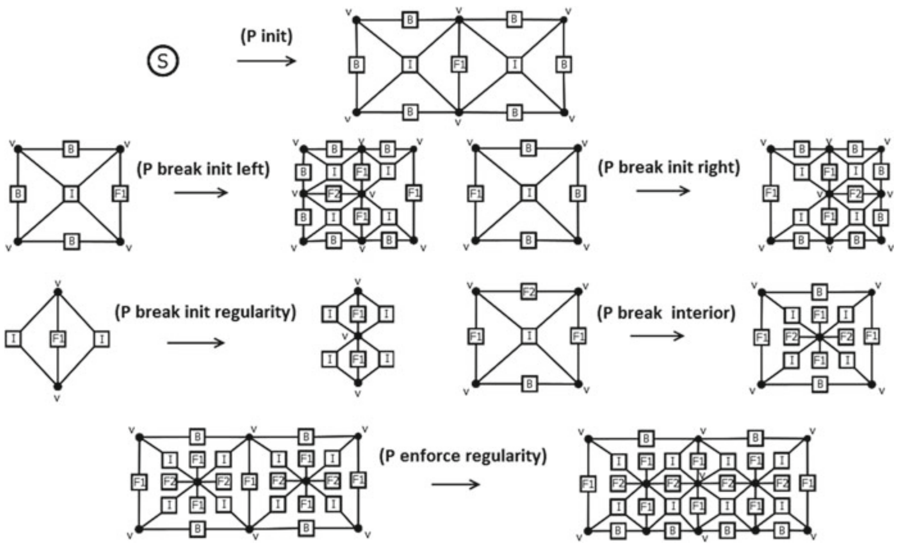


Fig. 7. Graph grammar productions generating the structure of the two finite element mesh h refined towards point singularity.

strucutre of the optimal elimination tree presented in Fig. 2, found by the optimization algorithm. The graph grammar model utilized in this paper is a new model based on the hypergraph transformations [16]. We start with graph grammar production responsible for generation of the structure of the computational mesh. These graph grammar productions are presented in Fig. 7. The **(P init)** graph grammar production generates the structure of the two initial finite elements. The finite element mesh vertices are denoted by v symbol, the finite element mesh edges are denoted by either B or $F1$ symbols, depending on their location on the boundary of the domain or inside the domain. The **(P break init left)** and **(P break init right)** graph grammar productions refine the mesh towards the central point singularity. The **(P break init regularity)** graph grammar productions enforces the mesh regularity rules on the common edge. After breaking of the initial mesh elements we can continue the refinement process towards the central point singularity by executing **(P break interior)** and **(P enforce regularity)** graph grammar productions. Since these graph grammar productions have to be executed in a sequence, the process of generation of the h refined computational mesh is linear, however the most expensive part of the algorithm is the generation of element frontal matrices and elimination of fully assembled nodes. This process can be executed in the logarithmic time. To achive that we introduce in Fig. 8 the graph grammar productions **(P inner elems alpha1–4)** and **(P strip alpha1–3)** modeling the generation of the frontal matrices related to particular levels of the mesh, followed by graph grammar productions **(P inner elems beta1–3)** modeling the elimination of the fully assembled nodes. These graph grammar productions can be executed

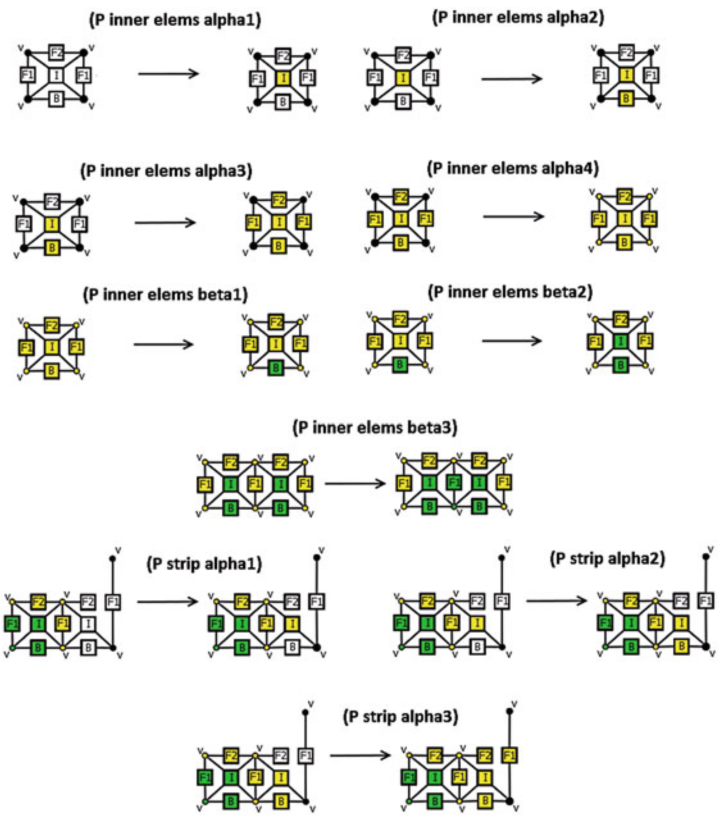


Fig. 8. Graph grammar productions performing aggregation (alpha) and elimination (beta) over mesh levels.

in parallel, over each level of the mesh at the same time. The **alpha** productions generate frontal matrices rows and columns related to particular nodes of the mesh, and the **beta** productions eliminate fully assembled nodes. In Fig. 8 we present only the subset of all the productions. Having the internal nodes eliminated from all the levels we can process the resulting interfaces in logarithmic time, using some additional graph grammar productions. In other words we browse the elimination tree from Fig. 4 level by level. The resulting computational cost of the solver algorithm is $O(logN)$ where N is number of nodes.

6 Numerical Results for Parallel Solver

Our heuristic algorithm for parallel solver delivers logarithmic computational cost, see Fig. 9. The solver has been executed on GeForce GTX 560 Ti graphic card with 8 multiprocessors, each one equiped with 48 cores. The total number of cores is equal to 384. The global memory on graphic card was 1024 MB. The comparison with parallel MUMPS executed on linux cluster shows that our

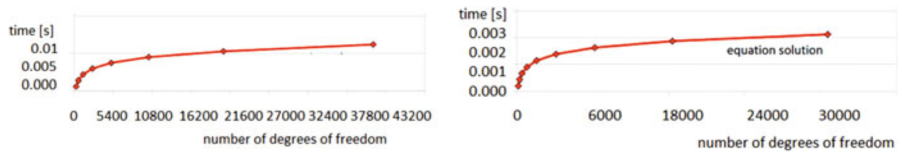


Fig. 9. Logarithmic computational cost of the parallel NVIDIA CUDA GPU solver executed for polynomial order of approximation $p = 1, 2$.

algorithm delivers better logarithmic time than parallel MUMPS solver with METIS library.

7 Conclusions and Future Work

In this paper we presented the optimization algorithm finding optimal elimination trees for multi-frontal direct solver working over the two dimensional meshes with point singularities. The optimization algorithm performed global search with dynamic programming algorithm resulting in the optimal elimination trees. The results of the optimisation procedure inspired us to construct heuristic algorithms for direct solver solution resulting in linear computational cost for sequential execution and logarithmic computational cost for parallel NVIDIA CUDA run. The obtained heuristic algorithms can be generalized for computational grids with many point singularities, and we can eliminated each point singularity in a linear cost in sequential mode and in logarithmic cost in parallel mode. The future work will involve generalization of the results into three dimensions, as well as development of the parallel algorithms performing reutilization of LU factorizations in order to speedup further the solver algorithm [14,15].

Acknowledgments. The work of MP and AP was supported by Polish National Science Center grant UMO-2012/07/B/ST6/01229. The work of PG was partly supported by The European Union by means of European Social Fund, PO KL Priority IV: Higher Education and Research, "Activity 4.1: Improvement and Development of Didactic Potential of the University and Increasing Number of Students of the Faculties Crucial for the National Economy Based on Knowledge, Subactivity 4.1.1: Improvement of the Didactic Potential of the AGH University of Science and Technology Human Assets", UDA POKL.04.01.01-00-367/08-00.

References

1. Amestoy, P.R., Duff, I.S., L'Excellent, J.-Y.: Multifrontal parallel distributed symmetric and unsymmetric solvers. Comput. Meth. Appl. Mech. Eng. **184**, 501–520 (2000)
2. Amestoy, P.R., Duff, I.S., Koster, J., L'Excellent, J.-Y.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM J. Matrix Anal. Appl. **23**(1), 15–41 (2001)

3. Amestoy, P.R., Guermouche, A., L'Excellent, J.-Y., Pralet, S.: Hybrid scheduling for the parallel solution of linear systems. Parallel Comput. **32**(2), 136–156 (2006)
4. Calo, V., Collier, N., Pardo, D., Paszyński, M.: Computational complexity and memory usage for multi-frontal direct solvers used in p finite element analysis. Procedia Comput. Sci. **4**, 1854–1861 (2011)
5. Demkowicz, L.: Computing with hp-Adaptive Finite Elements, vol. I. Chapman & Hall/Crc Applied Mathematics & Nonlinear Science (2006)
6. Duff, I.S., Reid, J.K.: The multifrontal solution of indefinite sparse symmetric linear systems. ACM Trans. Math. Softw. **9**, 302–325 (1983)
7. Duff, I.S., Reid, J.K.: The multifrontal solution of unsymmetric sets of linear systems. SIAM J. Sci. Stat. Comput. **5**, 633–641 (1984)
8. Geng, P., Oden, T.J., van de Geijn, R.A.: A parallel multifrontal algorithm and its implementation. Comput. Meth. Appl. Mech. Eng. **149**, 289–301 (2006)
9. Irons, B.: A frontal solution program for finite-element analysis. Int. J. Numer. Meth. Eng. **2**, 5–32 (1970)
10. Kuźnik, K., Paszyński, M., Calo, V.: Graph grammar-based multi-frontal parallel direct solver for two-dimensional isogeometric analysis. Procedia Comput. Sci. **9**, 1454–1463 (2012)
11. Karypis, G., Kumar, V.: MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, University of Minesota, http://www.cs.umn.edu/metis
12. Paszyńska, A., Paszyński, M., Grabska, E.: Graph transformations for modeling *hp*-adaptive finite element method with triangular elements. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part III. LNCS, vol. 5103, pp. 604–613. Springer, Heidelberg (2008)
13. Paszyński, M., Paszyńska, A.: Graph transformations for modeling parallel *hp*-adaptive finite element method. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 1313–1322. Springer, Heidelberg (2008)
14. Paszyński, M., Calo, V., Pardo, D.: A direct solver with reutilization of LU factorizations for h-adaptive finite element grids with point singularities. Comput. Math. Appl. **65**(8), 1140–1151 (2013)
15. Paszynski, M., Schaefer, R.: Reutilization of partial LU factorizations for self-adaptive *hp* finite element method solver. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part I. LNCS, vol. 5101, pp. 965–974. Springer, Heidelberg (2008)
16. Ślusarczyk, G., Paszyńska, A.: Hypergraph grammars in hp-adaptive finite element method. Procedia Comput. Sci. **18**, 1545–1554 (2013)

Parallel Efficiency of an Adaptive, Dynamically Balanced Flow Solver

Stanislaw Gepner(✉), Jerzy Majewski, and Jacek Rokicki

The Institute of Aeronautics and Applied Mechanics,
Warsaw University of Technology, Nowowiejska 24, 00-665 Warsaw, Poland
{sgepner,jmajewski,jack}@meil.pw.edu.pl
http://c-cfd.meil.pw.edu.pl

Abstract. Computations in Fluid Dynamics require minimisation of time in which the result could be obtained. While parallel techniques allow for handling of large problems, it is the adaptivity that ensures that computational effort is focused on interesting regions in time and space. Parallel efficiency, in a domain decomposition based approach, strongly depends on partitioning quality. For adaptive simulation partitioning quality is lost due to the dynamic modification of the computational mesh. Maintaining high efficiency of parallelization requires rebalancing of the numerical load. This paper presents performance results of an adaptive and dynamically balanced in-house flow solver. The results indicate that the rebalancing technique might be used to remedy to the adverse effects of adaptivity on overall parallel performance.

Keywords: Parallel cfd · Adaptation · Dynamic load balancing · Mesh refinement · Parallel efficiency · Super linear speed-up

1 Introduction

Large scale scientific simulations require high efficiency of parallelization in order to provide results in reasonable time. Commonly used domain decomposition approach requires the computational domain to be partitioned into subdomains, prior to the simulations. To maximize simulation performance, partitioning should minimize both processor idle time and the volume of interprocessor communication. This should be fulfilled throughout the simulation run time. To this end a range of partitioning methods and tools have been developed (see reference [5,11,12] for a survey). An example of an initial partitioning of a computational geometry is shown in Fig. 1.

Adaptive techniques allow for high resolution computations of localized phenomena (boundary layers, shock waves, etc), while limiting density of the discretization in less interesting regions. This increases the effectiveness with which computational infrastructure is used, as the necessary amount of resources is kept limited. Two alternative approaches to adaptivity are commonly used. The first, is based on global remeshing of the entire computational domain, by

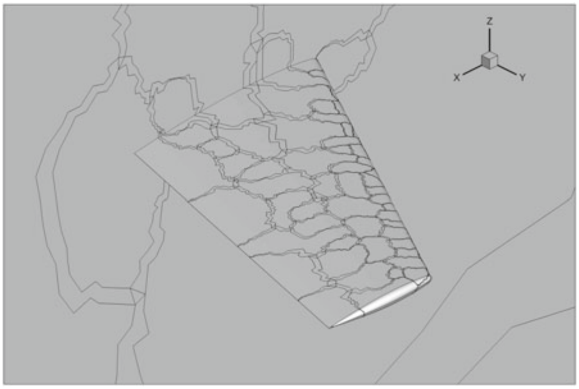


Fig. 1. Trace of the partitioning on the flow domain boundary around an Onera M6 wing geometry.

making use of an existing meshing techniques [1,3,6,13–15]. The second relies on application of local modifications to the computational mesh only in regions of interest [2,16,20,21].

The latter approach is particularly interesting for parallelization, since the resulting workflow consists of a series of localized mesh modifications. In this work a parallel adaptive algorithm described in [7,9] is used. The adaptive technique is based on edge splitting, employing predefined element division templates. Figure 2 presents both, the division templates for selected cases, and an example of a mesh resulting from couple of adaptive steps. Adaptive changes are driven by an error indicator based on estimated interpolation error (proportional to the Hessian of the current numerical solution). The approach is explained in detail in [1,2,6,13].

Adaptivity used during parallel computations might introduce significant modifications to the distribution of computational load. This has negative influence on the partitioning quality, and in consequence on the overall parallel effectiveness. Therefore a load balancing algorithm must be used together with adaptivity in a parallel simulation. (See [2,16,19].) A variation of such an approach, used in this work, has been described in [8] or [7]. The idea is to force migration of the computational load from excessively loaded partitions, to those possessing lesser workloads, as soon as the load balance indicator exceeds a certain, predefined threshold.

To measure partitioning quality (following [17]) this work employs a relation of the maximum numerical load, assigned to the computational processor, to the average numerical load to quantify the load-balance.

The load balance indicator is defined in the following way. Let the total computational effort, resulting from the problem being parallelized, be W. In homogeneous, parallel computational environment, an optimal load distribution is such that all computational processors are equally loaded. Therefore optimal processor load is $w_{opt} = W/p$, where p stands for the number of processors used.

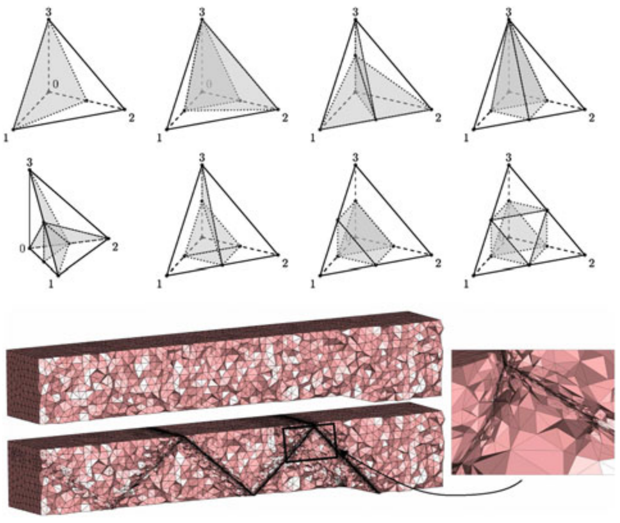


Fig. 2. Characteristic splitting templates (above) used for mesh refinement. Unadapted and adapted meshes for a 3D wedge test case (below).

Marking the actual numerical load of the i-th processor by w_i, and relating the maximum of processor load to the optimal value a simple load balance indicator is obtained:

$$\beta = \max_{0<i\leq p} \frac{w_i}{w_{opt}} \tag{1}$$

Using (1) to estimate the quality of a partitioning reflects a heuristic observation that waiting for the most loaded processor to finish its work is responsible for impeding parallel performance. Values, of the indicator β close to one, indicate a well balanced numerical load. The increasing imbalance in the load distribution will be reflected by higher values of the indicator. It should be noted, that (1) disregards the influence of communication overhead, resulting from given partitioning.

It is assumed, throughout this paper, that numerical load resulting from a grid based, finite element like method, is related to the number of degrees of freedom the considered system is represented by. Therefore units of total work W, are the Degrees of Freedom (DOFs).

Computational tests, both parallel and sequential have been carried out using a parallel machine equipped with 20 AMD Quad-Core 2354 (2.2 GHz) processors (in total up to 80 computational cores), with 2 GB of RAM memory per core.

2 Testing the Algorithmic Efficiency

The computational problem used for testing efficiency of parallelization comes form stationary simulation of a transonic flow through 2D Scramjet like geometry (Fig. 3). Problem is described by the set of Euler equations and discretized with

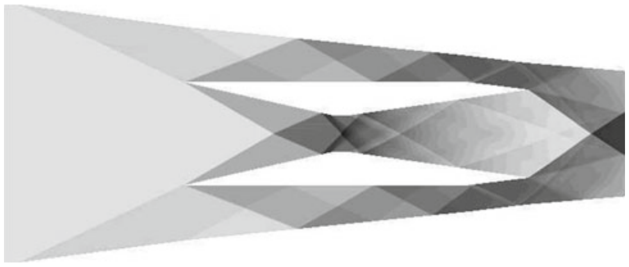


Fig. 3. Scramjet geometry and the resulting Mach filed contour.

the Residual Distribution method [4,18]. We use a sequence of homogeneous, isotropic, unstructured meshes resulting in problems of increasing numerical load, as they represent systems of increasing number DOFs. Flow parameters are selected as to reflect the situation of Mach 2 at the inflow. Selected flow regime results in a complex pattern of shock-waves to be created. Figure 3 shows test problem geometry and the resulting Mach field contours.

While, ultimately, it is the total computational time required to finalize the computation process, that is of interest, here we focus on measuring the speed at which computations are carried out. Therefore all performance estimates focus on measuring time required to perform an a-priori selected number of solver (or solver-communication for parallel case) cycles, rather than measuring time required to achieve a given convergence level.

Although, implicit schemes usually yield a better overall time performance, it might be difficult to effectively compare performance results. A single iteration of an implicit approach consists of a couple of sub steps. Amongst them are assembly of a linear system, application of a preconditioning method and finally the solution of the resulting system. Time performance of the whole process will be strongly influenced by each of the mentioned procedures. Moreover number of iterations necessary to achieve convergence of the linear system might vary as the solution progresses.

Considering the difficulties of an implicit method, this work is focused on estimating the computational performance of an explicit solver. We measure time necessary to complete a given number of computational cycles (including communication during synchronization sessions). All results presented are rescaled to correspond to 1000 solver cycles.

3 Sequential Algorithmic Efficiency

To accurately estimate parallel efficiency of a computational process one should compare results recorded for a sequential implementation. High parallel efficiency of any computational process might have its roots in suboptimal execution of the sequential code.

On the right Fig. 4 illustrates time required to perform a constant number of solver iterations, as a function of computational load, proportional to the

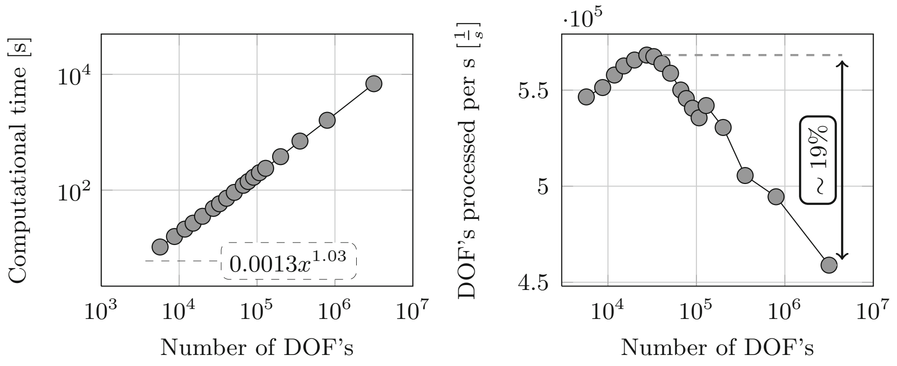


Fig. 4. Time required to perform 1000 explicit iterations (left). The computational performance measured as a number of DOF's processed per second in a single iteration, averaged over 1000 explicit iterations (right).

number of degrees of freedom. For comparison, a plot of a function fitted to the results through power regression, is shown. It should be observed that results indicate almost linear dependency of time on the numerical load applied.

Algorithmic efficiency, considered as speed at which computations are performed plotted versus the computational load is shown as well in Fig. 4 to the left. This illustrates the number of DOF's processed per second, averaged over 1000 solver cycles. The plot indicates a decrease in achievable computational speed, as the amount of work (related to the number of DOF's) increases.

4 Parallel Performance

4.1 Parallel Performance for Undisturbed Partitioning

Each of the test case meshes is used as a base for a parallel simulation. We present strong scaling results, in the form of speedup and execution time plots. Speedup S_p is defined by a ratio of time T_1, required for the computational task to be finished in a sequential computation (single processor), to time T_p, registered when computational problem is divided among p processors ($S_p = \frac{T_1}{T_p}$). Figure 5 illustrates parallel speedup resulting from parallelization of various test cases. Same results have been presented in Fig. 6, where registered execution times have been ploted. The results show, that parallelization of large problems results in better than theoretical speedup (super linear speedup). While parallelization of relatively small problems leads to an inefficient use of a parallel machine.

Performance results indicating super linear performance of some parallel algorithms have already been reported (see [10]) and results form utilization of fast cash memory for bigger part of the problem solved. Parallelization of a problem not only increases the number of processors, but also the amount of fast cash memory, which is available per subdomain. Additionally it should be noted that,

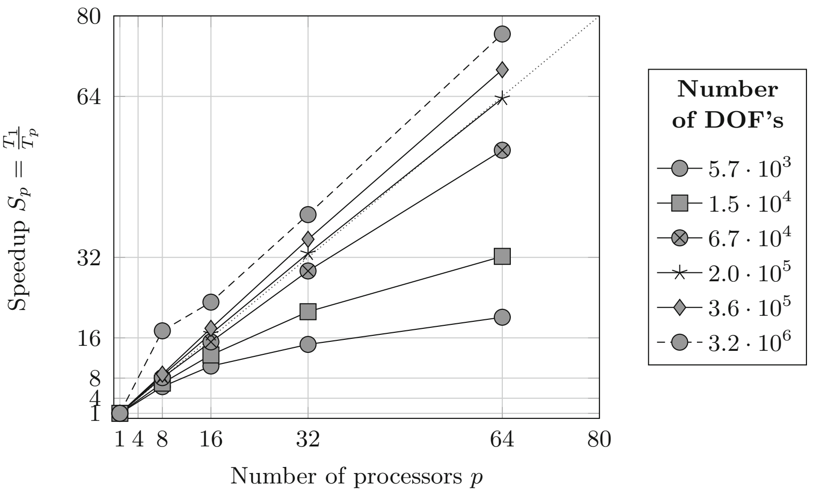


Fig. 5. Values of parallel speedup recorded for test cases with different number of degrees of freedom.

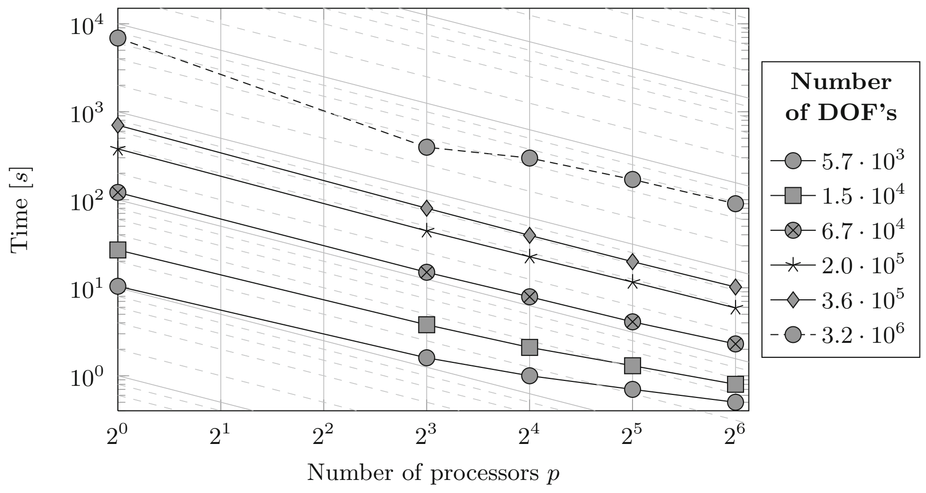


Fig. 6. Time required for the completion a parallelized problem registered for test cases with different number of degrees of freedom.

the sequential algorithmic performance tests, indicate a drop in computational speed, recorded for relatively large problems. Super-linear speed-up recorded for large test cases might be qualitatively correlated to the results presented in Fig. 4.

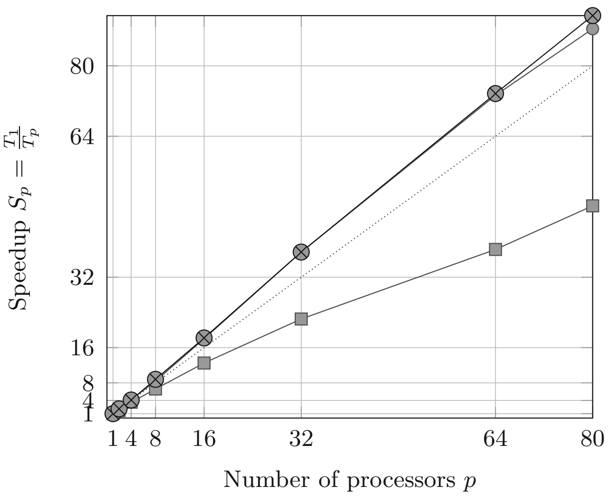


Fig. 7. Parallel speed-up registered for an adapted case with load balancing (—⊗—), compared with the adapted but not balanced (—■—) problem. For reference the unmodified, original case is shown (—●—).

4.2 Impact of Adaptivity on Parallel Performance

Modification to the mesh structure during a parallel run of a simulation will result in the sub optimal distribution of the numerical load. Such a modification is caused by an adaptive step, which results in changes to the computational mesh being used.

The influence of adaptivity on the parallel performance is presented using a test case originally resulting in $8.0 \cdot 10^5$ DOFs. Due to a single adaptive step number of degrees of freedom is increased to around $1.0 \cdot 10^6$. All parallel performance results for an adapted case are related to the sequential run on the adapted mesh.

Figures 7 and 8 show a speed-up plot and execution time plots respectively. Plots were obtained using parallel execution times registered for an adapted case. For comparison, a reference in the form of plots of speedup and scaling of the original ($8.0 \cdot 10^5$ DOFs), unadapted case are shown. Values of the load balance indicator, calculated with the use of formula 1, are shown for of the measurements presented in Fig. 8 in numerical boxes. For the well partitioned (initial case) it ranges from 1.0 to 1.05, while the adapted case yields results above 2. The load imbalance, resulting from an adaptive step, and following drop in parallel performance are substantial.

4.3 Parallel Performance with Dynamic Load Balancing

Results registered for the adapted case show necessity for rebalancing of the numerical load, should the available computational resources be used efficiently. A solution is to use the Dynamic Load balancing ([8] or [7]) strategy.

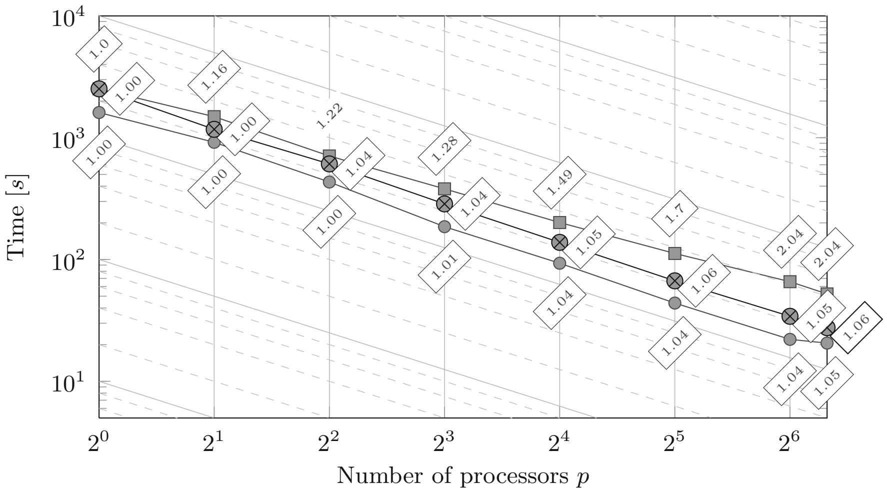


Fig. 8. Parallel scaling registered for an adapted case with load balancing (—⊗—), compared with the results for the unbalanced case (—■—). For reference the unmodified case is shown (—●—) Load balance indicator defined by Eq. 1 is provided for each of the cases (see the numerical boxes).

Figures 7 and 8 show plots or parallel speedup and scaling respectively. Measurements have been carried out for adapted and dynamically balanced test case used in Sect. 4.2. Again values of calculated balance indicator have been added to plots results in Fig. 8. Registered performance results for the dynamically balanced case relate to those registered for the original, undisturbed cases presented in Sect. 4.1.

5 Conclusions

Performance results of an explicit, adaptive, in-house flow solver have been presented. Both sequential and parallel performance has been considered. Results indicate high parallel performance, provided the numerical load is evenly distributed among sub domains. The influence of an adaptive process on the load distribution, and on to the parallel performance have been presented. Maintaining balance of the numerical load through a Dynamic Load Balancing repartitioning has been shown to recover high parallel performance of the adaptive application.

References

1. Alauzet, F., George, P.L., Mohammadi, B., Frey, P.J., Borouchaki, H.: Transient fixed point-based unstructured mesh adaptation. Int. J. Numer. Meth. Fluids **43**, 729–745 (2003)

2. Alauzet, F., Li, X., Seegyoung Seol, E., Shephard, M.S.: Parallel anisotropic 3D mesh adaptation by mesh modification. Eng. Comput. (Lond.) **21**(3), 247–258 (2006)
3. Castro-Dfaz, M.J., Hecht, F., Mohammadi, B., Pironneau, O.: Anisotropic unstructured mesh adaptation for flow simulation. Int. J. Numer. Meth. Fluids **25**, 475–491 (1997)
4. Deconinck, H., Sermeus, K., Abgrall, R.: Status of multidimensional upwind residual distribution schemes and applications in aeronautics. In: AIAA Conference Proceedings, pp. 2000–2328 (2000)
5. Devine, K.D., Boman, E.G., Heaphy, R.T., Hendrickson, B.A., Teresco, J.D., Faik, J., Flaherty, J.E., Gervasio, L.G.: New challenges in dynamic load balancing. Appl. Numer. Math. **52**(2–3), 133–152 (2005). (ADAPT '03: Conference on Adaptive Methods for Partial Differential Equations and Large-Scale Computation.)
6. Frey, P.J., Alauzet, F.: Anisotropic mesh adaptation for transient flows simulations. In: IMR, pp. 335–348 (2003)
7. Gepner, S., Majewski, J., Rokicki, J.: Dynamic load balancing for adaptive parallel flow problems. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009, Part I. LNCS, vol. 6067, pp. 61–69. Springer, Heidelberg (2010)
8. Gepner, S., Rokicki, J.: Dynamic load balancing for parallelization of adaptive algorithms. In: Kroll, N., Bieler, H., Deconinck, H., Couaillier, V., van der Ven, H., Sørensen, K. (eds.) ADIGMA. NNFM, vol. 113, pp. 327–338. Springer, Heidelberg (2010)
9. Gepner, S., Rokicki, J.: Investigation of parallel efficiency of an adaptive flow solver. Procedia Comput. Sci. **1**(1), 2673–2681 (2010). (ICCS 2010.)
10. Gustafson, J.L.: Fixed time, tiered memory, and superlinear speedup. In: Proceedings of the Fifth Distributed Memory Computing Conference DMCC5 (1990)
11. Hendrickson, B., Devine, K.: Dynamic load balancing in computational mechanics. Comput. Methods Appl. Mech. Eng. **184**(2–4), 485–500 (2000)
12. Hendrickson, B., Kolda, T.G.: Graph partitioning models for parallel computing. Parallel Comput. **26**(12), 1519–1534 (2000). (Graph Partitioning and Parallel Computing.)
13. Majewski, J.: An anisotropic adaptation for simulation of compressible flows. Math. Model. Anal. **7**, 127–134 (2002)
14. Majewski, J.: Anisotropic adaptation for flow simulations in complex geometries. In: 36th Lecture Series on Computational Fluid Dynamics/ADIGMA Course on HP-adaptive and HP-multigrid Methods. von Karman Institute for Fluid Dynamics (2009)
15. Majewski, J., Athanasiadis, A.: Anisotropic solution-adaptive technique applied to simulations of steady and unsteady compressible flows. In: Deconinck, H., Dick, E. (eds.) Computational Fluid Dynamics 2006 - Proceedings of the 4th International Conference on Computational Fluid Dynamics, ICCFD4, Ghent, Belgium, July 10–14, pp. 353–359. Springer, 2006
16. Park, Y.M., Kwon, O.J.: A parallel unstructured dynamic mesh adaptation algorithm for 3-d unsteady flows. Int. J. Numer. Meth. Fluids **48**, 671–690 (2005)
17. Rokicki, J., Żółtak, J., Drikakis, D., Majewski, J.: Parallel performance of overlapping mesh technique for compressible flows. Future Gener. Comput. Syst. **18**(1), 3–15 (2001)
18. Sermeus, K., Deconinck, H.: Solution of steady euler and navier-stokes equations using residual distribution schemes. In: 33rd Lecture Series on Computational Fluid Dynamics - Novel Methods for Solving Convection Dominated Systems (LS2003-05). von Karman Institute for Fluid Dynamics (2003)

19. Troyer, C., Baraldi, D., Kranzlmüller, D., Wilkening, H., Volkert, J.: Parallel grid adaptation and dynamic load balancing for a CFD solver. In: Di Martino, B., Kranzlmüller, D., Dongarra, J. (eds.) EuroPVM/MPI 2005. LNCS, vol. 3666, pp. 493–501. Springer, Heidelberg (2005)
20. Waltz, J.: Parallel adaptive refinement for unsteady flow calculations on 3d unstructured grids. Int. J. Numer. Meth. Fluids **46**, 37–57 (2004). doi:10.1002/fld.674
21. Zhu, Z., Wang, P., Tuo, S.: An adaptive solution of the 3-d euler equations on an unstructured grid. Acta Mech. **155**, 215–231 (2002). doi:10.1007/BF01176244

Modification of the Newton's Method for the Simulations of Gallium Nitride Semiconductor Devices

Konrad Sakowski[1,2](✉), Leszek Marcinkowski[2], and Stanislaw Krukowski[1,3]

[1] Institute of High Pressure Physics, Polish Academy of Sciences, ul. Sokolowska 29/37, 01-142 Warsaw, Poland
konrad@unipress.waw.pl

[2] Faculty of Mathematics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland

[3] Interdisciplinary Centre for Mathematical and Computational Modelling, Warsaw University, ul. Pawinskiego 5a, 02-106 Warsaw, Poland

Abstract. In this paper we present application of the Newton's method for simulation of gallium nitride semiconductor devices in the steady state.

The drift-diffusion model of carrier transport in the semiconductor material is used. It consists of three nonlinear elliptic differential equations. We present a backtracking strategy for the coupled Newton's method, which takes into account the specific nature of the drift-diffusion equations and improves convergence of the method.

Keywords: Drift-diffusion · van Roosbroeck equations · Gallium nitride · Coupled Newton method

1 Introduction

In modern times the computer simulation is an important tool of supporting process of design and development of new devices. In this paper we focus on numerical simulations of GaN-based semiconductor devices. A prime example of such a device is a blue laser.

In our simulations we use the drift-diffusion model of transport of charge carriers in the semiconductor material [7,10]. This model is called semi-classical, as it assumes classic spatial movement mechanisms and quantum mechanical carrier recombination. From the mathematical standpoint it is a system of three nonlinear elliptic differential equations, which is called the van Roosbroeck equations [8].

From the sixties of the 20th century this model is successfully used for simulation of semiconductor devices based on the variety of materials, for example silicon (transistors) and gallium arsenide (red lasers) [4]. Unfortunately the transition to the gallium nitride devices is not straightforward. The drift-diffusion model relies on profound simplifications of physical phenomena. This method yields fast simulations, but neglects some effects relevant for the gallium nitride heterostructures.

Our goal is to develop an algorithm capable of performing simulations, which is flexible enough to allow certain modifications of the standard drift-diffusion model. These modifications include: the piezoelectric effect, trap-assisted tunneling, interfacial charges, etc. Also the doping of the gallium nitride devices is much higher in comparison with the gallium arsenide, what leads to the convergence problems.

These modifications may be achieved by transformations of the underlying equations. Therefore we want to use a discretization of the van Roosbroeck equations, which is possibly general. Thus we have chosen the Composite Discontinuous Galerkin Method [1,3].

This paper is organized as follows. In Sect. 2 a differential model is briefly described. Then in Sect. 3 we propose a backtracking strategy for the Newton method. In Sect. 4 we compare it with standard backtracking based on residuum size and with the classic Newton method. Then we conclude in Sect. 5.

2 Drift-Diffusion Model

The drift-diffusion model describes relationship between the electrostatic potential and the charge carrier concentrations: electrons and holes [11,12]. Physical derivation of this model is beyond the scope of this work, therefore we will focus on the mathematical standpoint.

We start with the domain of our problem. Luminescent semiconductor devices are generally made of planar layers deposited one on another, which vary in composition of a semiconductor material or number of impurities. At opposite ends metal contacts are attached, where the current can be applied. If this is the case it flows through the device perpendicular to the deposited layers.

Since this flow is generally one-dimensional, we use a one-dimensional model to describe the device. This seems reasonable, as it improves the a speed of simulations greatly and reduces complexity of the problem. However, the results presented in this paper may be extended to two or three dimensions.

2.1 Differential Problem

The differential problem is to find functions $\psi, F_n, F_p : \Omega \to \mathbb{R}$, where Ω is an interval (polygon in 2D, polyhedron in 3D), such that

$$
\begin{aligned}
\nabla \cdot \Big(\varepsilon_0 \varepsilon(x) \nabla \psi(x)\Big) &= -qC(x, \psi, F_n, F_p), \\
\nabla \cdot \big(\mu_n(x) n(x, \psi, F_n) \nabla F_n(x)\big) &= qR(x, \psi, F_n, F_p), \\
\nabla \cdot \big(\mu_p(x) p(x, \psi, F_p) \nabla F_p(x)\big) &= -qR(x, \psi, F_n, F_p),
\end{aligned}
\tag{1}
$$

Since an algorithm presented in this paper does not rely on the particular form of the operators presented in (1), we omit the explanation of this system.

A detailed description may be found in [7,10], and an example of particular form used by us in physical applications may be found in [9]. We would like to mention that the coefficients $\varepsilon_0\varepsilon(x)$, $\mu_n(x)n(x)$ and $\mu_p(x)p(x)$ are all strictly positive in Ω. Therefore the drift-diffusion system in the formulation (1) is a system of nonlinear elliptic ordinary differential equations. The nonlinearity is both differential and algebraic. The above system is written in an unscaled form, with the potential ψ in volts and quasi-Fermi levels F_n, F_p in joules.

3 Algorithm

3.1 Discrete Problem

For discretization of the differential equations (1), we use the Composite Discontinuous Galerkin Method [3]. The algorithm we present is independent of a discretization method, therefore we assume that ψ, F_n, F_p are some approximations of the potential and quasi-Fermi levels respectively and they belong to a given discrete space. Let us denote

$$\psi = [\psi_1, \dots, \psi_J], \quad F_n = [F_{n,1}, \dots, F_{n,J}], \quad F_p = [F_{p,1}, \dots, F_{p,J}], \tag{2}$$

where the coefficients of ψ, F_n, F_p in a basis of the discrete space and J is a dimension of this space. We define ξ as

$$\xi = [\psi, F_n, F_p]. \tag{3}$$

We would like to use the Newton method to find an approximate solution of the discretized problem (1). Therefore let $a_\psi, f_\psi, a_n, f_n, a_p, f_p$ denote discrete problem operators (Composite Discontinuous Galerkin Method operators in our simulations) for left hand sides and right hand sides of the equations (1). Then let us define residual functions

$$\begin{aligned} G_{\psi,j}(\psi, F_n, F_p) &:= a_\psi(\psi, F_n, F_p, \varphi_{(j)}) - f_\psi(\psi, F_n, F_p, \varphi_{(j)}), \\ G_{n,j}(\psi, F_n, F_p) &:= a_n(\psi, F_n, \varphi_{(j)}) - f_n(\psi, F_n, F_p, \varphi_{(j)}), \\ G_{p,j}(\psi, F_n, F_p) &:= a_p(\psi, F_p, \varphi_{(j)}) - f_p(\psi, F_n, F_p, \varphi_{(j)}), \end{aligned} \tag{4}$$

where $\{\varphi_{(j)}\}_{j=1}^J$ is the base of the discrete space. Note that operators f_ψ, a_n, f_n, a_p, f_p are nonlinear in ψ, F_n, F_p, and linear in $\varphi_{(j)}$.

Then we define coupled residual function G as:

$$G(\xi) := [G_\psi(\xi), G_n(\xi), G_p(\xi)], \tag{5}$$

where

$$\begin{aligned} G_\psi(\xi) &:= [G_{\psi,1}(\xi), \dots, G_{\psi,J}(\xi)], \\ G_n(\xi) &:= [G_{n,1}(\xi), \dots, G_{n,J}(\xi)], \\ G_p(\xi) &:= [G_{p,1}(\xi), \dots, G_{p,J}(\xi)]. \end{aligned} \tag{6}$$

If $G(\xi)$ is zero, then ξ is a discrete solution. We may then pick some initial approximation ξ_0 and use the Newton's method to find the approximate solution.

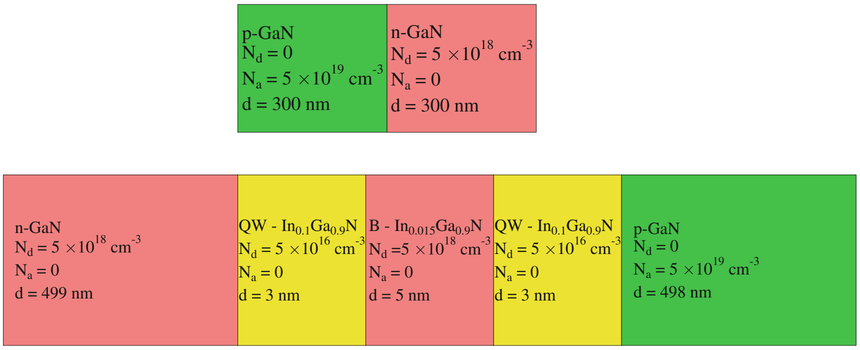

p-GaN
$N_d = 0$
$N_a = 5 \times 10^{19}$ cm^{-3}
d = 300 nm
n-GaN
$N_d = 5 \times 10^{18}$ cm^{-3}
$N_a = 0$
d = 300 nm
n-GaN
$N_d = 5 \times 10^{18}$ cm^{-3}
$N_a = 0$
d = 499 nm
QW - $In_{0.1}Ga_{0.9}N$
$N_d = 5 \times 10^{16}$ cm^{-3}
$N_a = 0$
d = 3 nm
B - $In_{0.015}Ga_{0.9}N$
$N_d = 5 \times 10^{18}$ cm^{-3}
$N_a = 0$
d = 5 nm
QW - $In_{0.1}Ga_{0.9}N$
$N_d = 5 \times 10^{16}$ cm^{-3}
$N_a = 0$
d = 3 nm
p-GaN
$N_d = 0$
$N_a = 5 \times 10^{19}$ cm^{-3}
d = 498 nm

Fig. 1. Schemes of devices used in simulations in this paper: a p-n diode and two quantum well heterostructure.

3.2 Problems with the Newton's Method

The Newton's method is very sensitive to the initial approximation. Unfortunately good initial approximations for the drift-diffusion model are available only for devices in so-called steady state, where the current is not connected. This is unsatisfactory, as the simulations mainly concern devices in operation. The idea is then to start a simulation from the steady state, and then gradually increase the voltage (bias) to the given value, which corresponds to changing of the boundary conditions. The sketch of the algorithm is then as follows:

```
$\xi_0$ := initial_approximation();
$i := 1$;
**for** bias:=0 **to** bias_max **step** bias_step **do**
  **while** $\|G(\xi_{i-1})\|$ is not small **do**
    $s_i := -[\mathrm{D}G(\xi_{i-1})]^{-1} G(\xi_{i-1})$;
    $\xi_i := \xi_{i-1} + s_i$;
    $i := i + 1$;
  **end while**
**end for**
```

In the above schema the *bias_max* denotes the target voltage of the device. This method may seem wasteful, as it needs additional outer loop for the bias. On the other hand, it is often advisable to perform the simulation for a range of voltages. For example simulation of the current-voltage characteristics, which may be then compared with experiments, involves calculating the current density as a function of the bias.

Unfortunately this straightforward algorithm does not perform well for the drift-diffusion simulations of GaN-based devices, as it diverges in many cases. The divergence as a result of overflows and underflows, which are easy to emerge due to functions $\exp(\pm\psi)$, $\exp(\pm F_n)$, $\exp(\pm F_p)$ present in the coefficients n, p (see [9]). Taking very small *bias_step* may be a remedy, but this increases the simulation time.

Therefore the next step could be to improve the convergence by taking some kind of backtracking method for the Newton's iteration [2]. The idea is then to scale the Newton's method step by some coefficient $0 < \lambda \leq 1$ in every iteration to ensure decrease of the norm $\|G(\xi)\|$ for a given norm $\|\cdot\|$. In our simulations we use $\mathbb{R}^n$ maximum norm. It can be shown [2] that if the Jacobian is nonsingular, then it is possible to find λ small enough to reduce the norm $\|G(\xi)\|$. When the approximation ξ will be close enough to the solution, $\lambda = 1$ is taken and the convergence would be as good as for the standard Newton's method.

The algorithm with very simple strategy of choosing λ may be written as follows (we omit the outer loop, as it does not change):

```
while $\|G(\xi_{i-1})\|$ is not small do
    $s_i := -[\mathrm{D}G(\xi_{i-1})]^{-1} G(\xi_{i-1})$;
    $\lambda_i := 1$;
    $\xi_i := \xi_{i-1} + s_i$;
    while $\|G(\xi_i)\| > \|G(\xi_{i-1})\|$ do
        $\lambda_i := \lambda_i / 2$;
        $\xi_i := \xi_{i-1} + \lambda_i \cdot s_i$;
    end while
    $i := i + 1$;
end while
```

The following example shows that this method is still disadvantageous. We present a simulation of a p-n GaN diode, which is a fairly simple device (Fig. 1). The result of 11th outer step of the simulation is shown on Fig. 2. Note the fluctuations of the functions F_n, F_p, which are erroneous from the physical point of view. However taking into account only the residuum size $\|G(\xi_i)\|$, this step converged perfectly. Initial approximation (from the previous bias-step) was

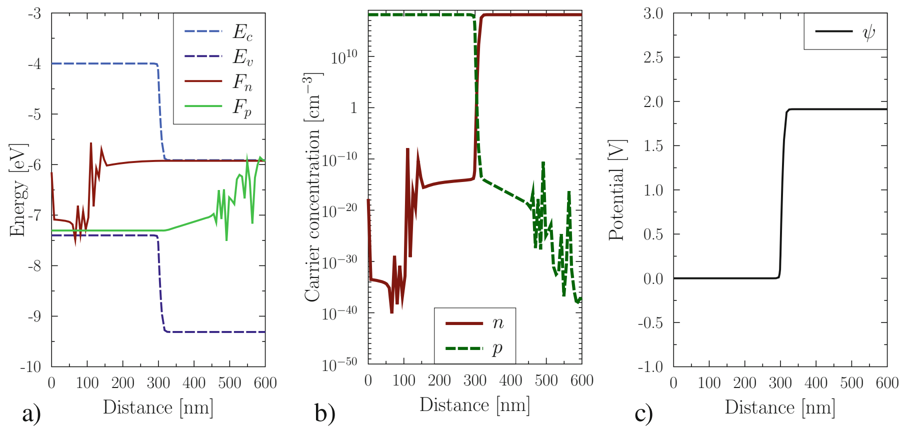


Fig. 2. An example of a simulation of the 600 nm GaN p-n diode simulated using the Newton method with linear backtracking. The bias is 1.38 V.

1.8×10^{11}, then four steps of the Newton method were performed with $\lambda = 1$, which reduced the residuum as follows: 3.4×10^6, 6.5×10^3, 2.7×10^{-2}, 6.6×10^{-4}. Thus the magnitude of the residuum was reduced by 14 orders, which is close to the machine precision. The backtracking was not even used.

Unfortunately this iteration diverged few bias-steps later, due to underflow. This is however not the case for every setting, sometimes the fluctuations vanish for a large bias and the algorithm does not break.

Still the question remains why such an nonphysical fluctuations may be present when the residuum is so small. The reason is that n and p are the coefficients of the second and third equation of (1), which formally correspond to F_n and F_p, respectively. On the left part of the device, where the fluctuations of F_n emerged, the coefficient n is very small, more than a 20 orders of magnitude smaller than on the other side of the device. Therefore this error is completely neglected due to the precision of the floating point arithmetic. Similar effect is observed for p.

Therefore we conclude that $\|G(\xi_i)\|$ is not a good measure of quality of the solution, as the residuum is not of the same order on the whole domain. We must therefore search for better indicator of the status of the approximation.

3.3 Modification of Newton's Method

The weakness of the backtracking algorithm presented in previous section is a lack of estimate of the quality of approximations, as the residuum is suitable only for some parts of a device. In this section we would like to show how to get such an estimate which gives good results on the whole domain. Inspired by [5], we would like to rewrite the problem in the Banach iteration scheme.

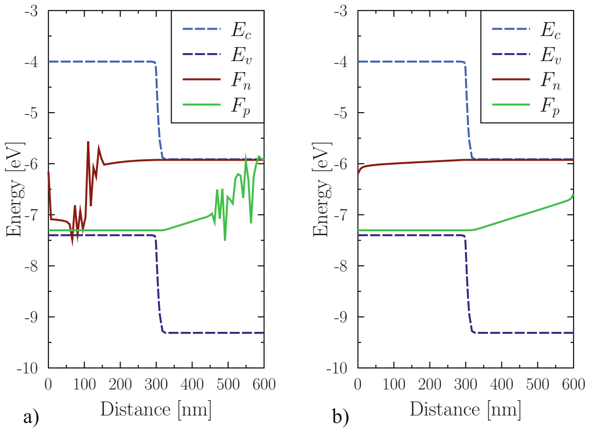


Fig. 3. Comparison of simulation results of the 600 nm GaN p-n diode simulated using: (a) the Newton method with linear backtracking, (b) the Newton method with our modification. The bias is 1.38 V.

Let $(\psi_0, F_{n,0}, F_{p,0})$ be some initial approximation. Let us define function T as

$$\xi_i = T(\xi_{i-1}) \tag{7}$$

where $\xi_i = (\psi_i, F_{n,i}, F_{p,i})$ is a solution of the discrete version of a following system of differential equations

$$\begin{aligned} \nabla \cdot \Big(\varepsilon_0 \varepsilon \nabla \psi_i\Big) &= -qC(\psi_{i-1}, F_{n,i-1}, F_{p,i-1}), \\ \nabla \cdot \big(\mu_n n(\psi_{i-1}, F_{n,i-1}) \nabla F_{n,i}\big) &= qR(\psi_{i-1}, F_{n,i-1}, F_{p,i-1}), \\ \nabla \cdot \big(\mu_p p(\psi_{i-1}, F_{p,i-1}) \nabla F_{p,i}\big) &= -qR(\psi_{i-1}, F_{n,i-1}, F_{p,i-1}), \end{aligned} \tag{8}$$

where $\xi_{i-1} = (\psi_{i-1}, F_{n,i-1}, F_{p,i-1})$. If $\xi_i = \xi_{i-1}$, then ξ_i is a solution of the discrete problem. Note that (8) is a system of three independent linear differential equations, so $T(\xi_{i-1})$ may be computed easily.

We do not aim at finding a solution by Banach iteration for T, as generally it is not a contraction. We would like to use T for estimate of the quality of solutions in a following manner. Let us define H as

$$H(\xi) := T(\xi) - \xi. \tag{9}$$

Assume that ξ_{i-1} is close to the solution. Then $\xi_i = T(\xi_{i-1}) \approx \xi_{i-1}$. Unlike n and p, functions ψ, F_n and F_p have the same order of magnitude on the whole domain, and they are of similar order under appropriate choice of units (ψ in volts; F_n, F_p in electronvolts). Therefore elements of the vector $H(\xi_{i-1})$ do not vary by orders of magnitude and $\|H(\xi_{i-1})\|$ may be used as an estimate of an approximation ξ_{i-1}, because it has no drawbacks of $\|G(\xi_{i-1})\|$.

Therefore we propose the following modification of the inner loop:

```
**while** $\|H(\xi_{i-1})\|$ is not small **do**
  $s_i := -[\mathrm{D}G(\xi_{i-1})]^{-1} G(\xi_{i-1})$;
  $\lambda_i := 1$;
  $\xi_i := \xi_{i-1} + s_i$;
  **while** $\|H(\xi_i)\| > (1+c)\|H(\xi_{i-1})\|$ **do**
    $\lambda_i := \lambda_i / 2$;
    $\xi_i := \xi_{i-1} + \lambda_i \cdot s_i$;
  **end while**
  $i := i + 1$;
**end while**
```

Generally for $c = 0$ this modification tends to minimize $\|H(\xi_i)\|$. Our observations show that it is often favorable to allow controlled growth of $\|H(\xi_i)\|$ by setting $c > 0$.

To illustrate usefulness of the function H, we will revisit our example from the previous section (Fig. 2). The nonphysical solution had a residuum norm 6.6×10^{-4}. However, if we measure $\|H(\xi)\|$ for this approximation, we obtain 1.1×10^{14}. Therefore without question $\xi \not\approx T(\xi)$, which is the information we expect to gain.

We have therefore repeated this simulation, using the algorithm proposed in this section. The result is presented on the Fig. 3(b). In this case there are no fluctuations of F_n, F_p, $\|G(\xi)\| \approx 3.2 \times 10^{-5}$ and $\|H(\xi)\| \approx 4.6 \times 10^{-6}$. Therefore the residuum is similar as for the nonphysical case, but the latter value is much lower, which corresponds to better quality of this approximation.

4 Comparison

As we pointed out in Sect. 3, the backtracking strategy proposed in this paper may prevent divergence and lead to the approximations, which are physically more favorable. Still we would like to show that it is also more efficient than the standard Newton method in terms of iteration number and computational time.

Therefore we compare simulation results for a two quantum well heterostructure presented on Fig. 1. We take into account the classic Newton method, backtracking linesearch [2], and our backtracking strategy. Simulations account for radiative recombination, Shockley-Read-Hall recombination with trap-assisted tunneling [4], ionization of impurities and piezoelectric effect.

A goal of these simulations were to find an approximate solution of the drift-diffusion equations for bias 4 V. We have to point out that it is not feasible to compute the solution for nonzero bias with the Newton method alone, or using the inner loop of the presented algorithms, as the initial approximations are only available for so-called steady state of a device, when bias is zero. So to perform our simulations we set `bias_max` to 4 V. Every consecutive solution is used as initial approximation for next inner loop. It is generally not a waste, as they are also used to compute a IV characteristic on [0, `bias_max`], which are used to compare results with physical experiments in real simulations. To obtain a fine IV characteristic, it is enough to have 10–20 steps, as it generally should not fluctuate much.

Since the Newton method is sensitive to an initial approximation, generally more steps should improve the convergence of the considered methods (number of steps = 1 + `bias_max/bias_step`). However, too much steps would increase the total iteration number and it is not very beneficial to the IV characteristic.

Results of these simulations are presented in Table 1. For each method we performed few simulations with an outer iteration number varying from 21 to 331. Every method considered in this study diverged if the number of steps was below 21.

In this setting, the most efficient was the Newton method with our backtracking strategy. The simulation took 284 s and 174 iterations in 21 steps. Next one was the linear backtracking on $\|G(\xi)\|$, with a stop condition on $\|H(\xi)\|$, which took 482 s and 377 iterations in 101 steps. For lower number of steps, the latter method generally did not return satisfactory results. In comparison, results of standard linear backtracking were acceptable for 161 steps (714 iterations, 898 s). Note that the classic Newton method with no backtracking was more efficient, so the linear backtracking did not help.

Table 1. Comparison of efficiency of our modification with linear backtracking and the classic Newton method, with stop conditions imposed on $\|G(\xi)\|$ or $\|H(\xi)\|$. In this table we present an outer iterations number (steps), total iteration number of the Newton method, computation time and average number of iterations per one step. Simulations were performed on a standard desktop PC. In every simulation, the Newton iteration steps were computed for $G(\xi)$

Our backtracking strategy allows slight increase of $\|H(\xi_i)\|$				
Steps	Iter.	Iter./steps	Time (s)	Result
21	174	8.29	284	Good
41	227	5.54	360	Good
91	378	4.15	610	Good
101	409	4.05	666	Good
161	566	3.52	919	Good
331	908	2.74	1516	Good

Linear backtracking stop condition on $\|G(\xi_i)\|$				
Steps	Iter.	Iter./steps	Time (s)	Result
21	—	—	—	Diverged
41	317	7.73	377	Nonphysical
91	544	5.98	670	Nonphysical
101	465	4.60	581	Nonphysical
161	714	4.43	898	Good
331	870	2.63	1189	Good

Linear backtracking stop condition on $\|H(\xi_i)\|$				
Steps	Iter.	Iter./steps	Time (s)	Result
21	—	—	—	Diverged
41	234	5.71	288	Nonphysical
91	354	3.89	452	Nonphysical
101	377	3.73	482	Good
161	519	3.22	676	Good
331	870	2.63	1190	Good

Classic Newton method stop condition on $\|H(\xi_i)\|$				
Steps	Iter.	Iter./steps	Time (s)	Result
21	—	—	—	Diverged
41	—	—	—	Diverged
91	—	—	—	Diverged
101	378	3.74	484	Nonphysical
161	510	3.17	665	Good
331	870	2.63	1189	Good

Classic Newton method stop condition on $\|G(\xi_i)\|$				
Steps	Iter.	Iter./steps	Time (s)	Result
21	—	—	—	Diverged
41	—	—	—	Diverged
91	—	—	—	Diverged
101	398	3.94	496	Nonphysical
161	612	3.80	762	Good
331	1197	3.62	1497	Good

Generally setting the number of outer steps to a number high enough leads to convergence of every tested method. Then the number of iterations become similar, as methods need not backtrack due to good initial approximations.

Our simulations also reveals that imposing a stop condition on $\|H(\xi)\|$ alone leads to slightly better efficiency, but an improvement is not so profound.

One more possibility, which is not discussed in this paper, is application of the Newton method directly to a problem $H(\xi) = 0$. However it that case Jacobian DH is dense despite of finite element method discretization [6], and the method is inefficient.

5 Conclusions

In this paper the novel backtracking strategy for the Newton method for the drift-diffusion model is shown. This modification improves convergence of the coupled Newton method applied to a discrete van Roosbroeck equation system.

We have shown that the standard backtracking strategies for the Newton method, which base on a residuum size, are error-prone due to large fluctuations of the coefficients of the equations. Such a situation is common for the gallium nitride semiconductor devices and the algorithm should take it into account.

The presented algorithm does not involve scaling of the unknown functions on a differential level and it does not rely on the discretization. Therefore it allows straightforward modifications of the underlying equations. Such modifications may account for new recombination mechanisms, ionization of traps, interfacial charges, etc. Examples of applications of our algorithm to real devices may be found in [9].

Acknowledgements. The research was supported by European Union within European Regional Development Fund, through grant Innovative Economy (POIG.01.01.02-00-008/08).

References

1. Arnold, D.N., Brezzi, F., Cockburn, B., Marini, L.D.: Unified analysis of discontinuous Galerkin methods for elliptic problems. SIAM J. Numer. Anal. **39**(5), 1749–1779 (2001)
2. Dennis, J.E., Schnabel, R.B.: Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentiece-Hall Inc., Englewood Cliffs (1983)
3. Dryja, M.: On discontinuous Galerkin methods for elliptic problems with discontinuous coefficients. Comput. Meth. Appl. Math. **3**(1), 76–85 (2003)
4. Hurkx, G., Klaassen, D., Kmuvers, M.: A new recombination model for device simulation including tunneling. IEEE Trans. Elect. Dev. **39**, 331–338 (1992)
5. Jerome, J.W., Kerkhoven, T.: A finite element approximation theory for the drift diffusion semiconductor model. SIAM J. Numer. Anal. **28**(2), 403–422 (1991)
6. Kerkhoven, T., Saad, Y.: On acceleration methods for coupled nonlinear elliptic systems. Numer. Math. **60**, 525–548 (1992)
7. Markowich, P.A., Ringhofer, C.A., Schmeiser, C.: Semiconductor Equations. Springer, Wien (1990)
8. Polak, S.J., den Heijer, C., Schilders, W.H.A.: Semiconductor device modelling from the numerical point of view. Int. J. Numer. Meth. Eng. **24**, 763–838 (1987)
9. Sakowski, K., Marcinkowski, L., Krukowski, S., Grzanka, S., Litwin-Staszewska, E.: Simulation of trap-assisted tunneling effect on characteristics of gallium nitride diodes. J. Appl. Phys. **111**, 123115 (2012)
10. Selberherr, S.: Analysis and Simulation of Semiconductor Devices. Springer, Wien (1984)
11. Sze, S., Ng, K.: Physics of Semiconductor Devices. Wiley-Interscience, Berlin (2006)
12. Wilkes, P.: Solid State Theory in Metallurgy. Cambridge University Press, Cambridge (1973)

Numerical Realization of the One-Dimensional Model of Burning Methanol (Cluster Version)

Krzysztof Moszyński(✉)

Faculty of Mathematics, University of Warsaw, Banacha 2, 02-097 Warszawa, Poland
kmoszyns@mimuw.edu.pl
http://www.mimuw.edu.pl

Abstract. This program developed for IBM Blue Gene/P is based on the so-called method of time splitting. While currently the MPI standard is used, in the future a version utilizing the OpenMP standard with shared memory will be developed as well.

Keywords: Time splitting · Schur system · Parallel processing

1 Introduction

This mathematical model of methanol burning in oxygen in a chemical reactor was recently proposed by Jerzy Bałdyga (Warsaw University of Technology, Poland) and Marek Burnat (University of Warsaw, Poland). Corresponding chemical reaction can be written as follows:

$$\boldsymbol{2CH_3OH + 3O_2 \rightarrow 2CO_2 + 4H_2O}.$$

In equations describing this process the functions $\rho_1,\ \rho_2,\ \rho_3,\ \rho_4$ are so called α-mass densities of the corresponding components $\mathbf{CH_3OH}$, $\mathbf{O_2}$, $\mathbf{CO_2}$, and $\mathbf{H_2O}$. For the usual **Euler densities** we have

$$\varrho_i(t,x) = \kappa_i \int_{\mathcal{A}} \rho_i(t,x,\alpha)d\alpha,\ i = 1,2,3,4, \tag{1}$$

where $\mathcal{A}$ is the set of velocities $\alpha,\ \beta,\ \cdots$ admitted in the model, and κ_i are constants.

2 Equations of the Model

The model is defined by the following equations:

$$\rho_{i_t} + \alpha\rho_{i_x} - \nu\rho_{xx} - K_i = \frac{1}{\kappa_i}\int_{\mathcal{A}} M_i(\cdot,\alpha,\beta)d\beta,\ i = 1,2,3,4 \tag{2}$$

This work is supported by grant Nr. G33-10 of the ICM - Interdisciplinary Center for Mathematical and Computational Modeling of the University of Warsaw.
Author thanks R. Wyrzykowski for his valuable Remarks.

where $t \in [0,T]$ is time, x is a point in the reactor $\Omega = [0,L]$, $\alpha \in \mathcal{A} = [-\gamma,\gamma]$. Functions K_i describe **kinetics of the chemical reactions** and are defined as follows:

$$\begin{aligned} K_1(\rho_1,\rho_2) &= -C_1(C_2\rho_1)^{q_1}(C_2\rho_2)^{q_2} < 0, \\ K_2 &= const. < 0, \\ K_3 &= -K_1 > 0, \\ K_4 &= -2K_1 > 0. \end{aligned} \tag{3}$$

where C_1, C_2, q_1, q_2 are positive constants, while K_2 is a negative. Moreover, ν is a small positive constant of diffusion, and the right hand side of Eq. (2) is the **mixer**. Let us define now this mixer. Put:

$$r(d) = \begin{cases} \dfrac{-d}{1+d} & \text{for} \quad d \geq 0 \\ \dfrac{-d}{1-d} & \text{for} \quad d < 0 \end{cases}$$

$$d_i = |\beta|\rho_i(t,x,\beta) - |\alpha|\rho_i(t,x,\alpha)$$

$$M_i(t,x,\alpha,\beta) = \begin{cases} r(d_i)\rho(t,x,\alpha) & \text{for} \quad d_i \geq 0 \quad \alpha\beta \geq 0 \\ r(d_i)\rho(t,x,\beta) & \text{for} \quad d_i < 0 \quad \alpha\beta \geq 0 \\ \eta r(d_i)\rho(t,x,\alpha) & \text{for} \quad d_i \geq 0 \quad \alpha\beta < 0 \\ \eta r(d_i)\rho(t,x,\beta) & \text{for} \quad d_i < 0 \quad \alpha\beta < 0 \end{cases} \tag{4}$$

with a chosen constant $\eta \in [0,1)$. Observe that in this way the **mixers** are controlled by the differences d_i of α - **impulses** for $i = 1,2,3,4$.

Equations (2)–(4) are completed by initial conditions on $[0,L]$ and Dirichlet boundary conditions at $x = 0$ and $x = L$. Time variable t runs over the interval $[0,T]$. Since certain initial conditions contain jumps, a spatial smoothing procedure may be applied.

3 Numerical Realization on Cluster

Equations (2) are approximated by finite differences on 2D rectangular grid $\mathcal{G}$, with the basic time step $\tau = T/N$ and the space step $h = L/M$. Let $t^n = n\tau$ and $x_k = kh$. Hence $(t^n, x_k) \in \mathcal{G}$ for $0 \leq n \leq N$ and $0 \leq k \leq M$. Assume that lp processors will be used, enumerated as usual: $s = 0, 1, \cdots, lp-1$. We assume also that at any time-level t^n, the processor of the number s will process all four equations (2), but on the part of the grid $\mathcal{G}$ containing only the points with x_k, such that $sR \leq k \leq (s+1)R - 1$, where R is a chosen integer. Observe that the above conditions imply that the number M (of subintervals on the x-axis) has to satisfy the condition $M = lp \cdot R - 1$.

Such a decomposition strategy is motivated by nasty properties of **mixers**, which, for any fixed pair (t,x), have to run over all ordered pairs (α,β), executing very large number of operations. Decomposition across the set $\mathcal{A}$ of admitted velocities could force us to use the time consuming inter-processor data sending.

Our goal is to make this program as fast as possible. We hope, we can succeed with this strategy.

Here, approximation of Eq. (2) is mainly based on the so called **time splitting method**. This is a very old method, probably firstly invented and applied in years 60s of 20-th century, by **D.W. Paceman and H.H. Rachford jr.** For large bibliography, see for example [1]. This method, suitable mainly for evolution problems, consists in dividing the numerical problem into some parts and solving these parts independently at each time step (or at parts of each step). These partial solutions are joined by initial conditions. Such a method may allow to split, at each time step, a complex numerical problem into several simple sub-problems, which are solved independently. Quality of such kind of approximation was discussed by many authors. In general, algorithms of this kind may give approximation of the order $O(\tau)$, but in some cases $O(\tau^2)$ is possible. The problem somewhat similar to this considered here, is discussed in [2].

4 Splitting of Equation of the Model

We start with splitting of Eq. (2). The following example explains the general scheme of the time splitting method used here. Suppose, we want to solve an equation of the form

$$u_t(t) = f(u) + g(u), \quad u(0) = a_0, \tag{5}$$

where, if necessary, definitions of functions f and g may contain also definitions of boundary conditions. Assume that on the t-axis there are defined grid points $t^n,\ n = 0, 1, 2, \cdots$ with a constant ("basic") time-step τ, $t^{n+1} = t^n + \tau$. Starting at the grid point t^n, with initial condition $u(t^n) = a_n$, we want to make one step to the grid point t^{n+1}. Applied approximation consists in the following splitting:

$$v_t(t) = f(v),\ v(t^n) = u(t^n) = a_n,\ a_{n+1} = v(t^{n+1})$$

$$w_t(t) = g(w),\ w(t_n) = a_{n+1},\ u(t^{n+1}) \approx w(t^{n+1}),$$

and so on. Above splitting scheme can be graphically presented as follows

$$\begin{tikzcd}
\boldsymbol{t^{n+1}} & \boldsymbol{v(t^{n+1})} \arrow[drr] & & \boldsymbol{w(t^{n+1}) \approx u(t^{n+1})} \\
\boldsymbol{t^n} & \boldsymbol{a_n} \arrow[u, "\boldsymbol{v}"] & & \boldsymbol{a_{n+1}} \arrow[u, "\boldsymbol{w}"]
\end{tikzcd} \tag{6}$$

For analysis of this approximation procedure see, for example [2].

We split system (2) into the following sub-systems.[1]

1. **Pure linear transport**

$$u_{it}(t, x, \alpha) + \alpha u_{ix}(t, x, \alpha) = 0,\ u_i(t^n, x, \alpha) = \rho_i(t^n, x, \alpha),\ i = 1, 2, 3, 4. \tag{7}$$

[1] Similar splitting was applied with success to another system of equations.

We consider separately cases $\alpha \geq 0$ with the left Dirichlet condition, and $\alpha < 0$ with right Dirichlet condition.

2. **Diffusion**

$$v_{it} = \nu v_{ixx}, \; v_i(t^n, x, \alpha) = u_i(t^{n+1}, x, \alpha), \; i = 1, 2, 3, 4. \tag{8}$$

with Dirichlet conditions on both sides.

3. **Nonlinear terms**

$$w_{it} = K_i(w_1, w_2) + \int_{\mathcal{A}} M_i(t, x, w_i, \alpha, \beta) d\beta, \tag{9}$$

$$w_i(t^n, x, \alpha) = v_i(t^{n+1}, x, \alpha), \; i = 1, 2, 3, 4.$$

According to schema (6), each subsystem has to be solved for $t \in [t^n, t^{n+1}]$, and $\rho_i(t^{n+1}, x, \alpha) \approx w_i(t^{n+1}, x, \alpha)$.

5 Algorithms for Subsystems

We introduce notation: $\lambda = \tau/h, \; \mu = \tau/h^2$;
$e_1 = [1, 0, \cdots]^T, \; e_R = [0, \cdots, 1]^T$ for R-dimensional column vectors;
$u_k^n = u(t^n, x_k)$ for function u defined on the grid $\boldsymbol{\mathcal{G}}$.

1. **Linear transport: $\boldsymbol{u_t + \alpha u_x = 0}$**
We use so called **box scheme** to approximate this equation. In case when $\boldsymbol{\alpha \geq 0}$ this (explicit, unconditionally stable!) scheme is as follows:

$$\begin{aligned} \boldsymbol{a u_k^{n+1} + b u_{k-1}^{n+1}} &= \boldsymbol{b u_k^n + a u_{k-1}^n, \; k = 1, 2, \cdots, M-2,} \\ \boldsymbol{u_M^{n+1}} &= \boldsymbol{u_{M-1}^n,} \end{aligned} \tag{10}$$

where $\boldsymbol{a = 1 + \lambda\alpha, \; b = 1 - \lambda\alpha}$.
Since we work in parallel, it is better to formulate this problem as the following system of linear equations with two-diagonal matrix:

$$\begin{bmatrix} a & \cdot & \cdot & \cdot & \cdot \\ b & a & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & b & a \end{bmatrix} x = f$$

where f is the vector defined by the right hand side of schema. In order to adjust the system to parallel processing on lp processors, we write it down in block form with quadratic blocks of size $R \times R$:

$$\begin{bmatrix} A & \cdot & \cdot & \cdot & \cdot \\ B & A & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & B & A \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \cdot \\ x_p \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \cdot \\ f_p \end{bmatrix}$$

where $p = lp - 1$. Observe that for $s = 0, 1, \cdots, lp - 1$ all diagonal blocks are two-diagonal, with a at main diagonal, and b at under-diagonal. All under-diagonal blocks are of the form $B = be_1e_R^T$. Parallel algorithm is as follows:

(a) In processor of number $s = 0, 1, \cdots, lp - 1$, two systems of linear equations with matrix A are solved: first, with right hand side vector f_s, and second, with right hand side vector e_1. Elements f_0 and/or f_M contain additional terms coming from Dirichlet boundary conditions. Observe that matrix of system is triangular, with only main diagonal and one sub-diagonal, hence resolution of these two systems of size R is very simple. Let the corresponding solutions be $\tilde{x}_s$, and w, respectively, and put $z_s = e_R^T x_s$ for $s = 0, \cdots, lp - 1$. It is easy to see that

$$x_0 = \tilde{x}_0,$$

$$x_s = \tilde{x}_s - bwz_{s-1}, \; s = 1, \cdots, lp - 1, \tag{11}$$

From the above definition it follows that numbers

$$z_s, \quad s = 1, 2, \; \cdots, lp - 1$$

satisfy the following two-diagonal system

$$\Sigma z = \begin{bmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ g & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & g & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \cdot \\ z_{lp-1} \end{bmatrix} = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \cdot \\ \tilde{x}_{lp-1} \end{bmatrix}, \tag{12}$$

where $g = be_R^T w$. This is **Schur system** of our problem. Let us observe that matrix Σ can be formed in each processor.

(b) In order to solve system (10) in parallel way in lp processors, we have first to solve **Schur system** (12) in each processor. To do that, we have to form a **lp**-dimensional vector $F_s = [F(0)_s, \cdots, F(lp-1)_s]^T$ with coordinates

$$F(j)_s = \begin{cases} 0 & \text{for } j \neq s \\ \tilde{x}_s & \text{for } j = s \end{cases}$$

in s-th processor.

Using now (only once!) **MPI** routine **ALLREDUCE with option "SUM"** [5,6] for vectors $F_s, \; s = 0, \cdots, lp-1$, we get in each processor the complete right hand side vector $\tilde{x}$ of the **SCHUR system**. Now it remains only to solve this **SCHUR system**, and we get the whole vector $z = [z_1, \cdots, z_{lp-1}]^T$ in each processor.

(c) The last step is to use s-th equation of formula (11) in s-th processor. In such a way we get s-th coordinate x_s of the solution in s-th processor.

Remark. Method described above is a two diagonal version of algorithm **"Divide and Conquer"** first defined and used for three diagonal systems by **Stefan Bondeli, ETH Zürich**, in 1990, see [3]. Tests for the **"Divide and Conquer" algorithm** [3] for various three-diagonal linear systems of size up to 10^6 and on various numbers of processors up to

512 were done on **IBM Blue Gene/P "Notos"** in ICM (version of 4 processors per node). Time measured was very nearly inversely proportional to the number of processors used. Hence this result may be considered as very satisfactory.

2. **Diffusion**
For approximation of equation

$$u_t = \nu u_{xx}$$

with initial condition and Dirichlet boundary conditions at both sides, **Crank - Nicolson** implicit scheme was applied:

$$du_0^{n+1} - au_1^{n+1} = d_1 u_0^n + au_1^n$$

$$du_k^{n+1} - a(u_{k-1}^{n+1} + u_{k+1}^{n+1}) = d_1 u_k^n + a(u_{k-1}^n + u_{k+1}^n),\ k = 1, \cdots, M-1$$

$$du_M^{n+1} - au_{M-1}^{n+1} = d_1 u_M^n + au_{M-1}^n,$$

where $d = (1 + \mu\nu)$, $d_1 = (1 - \nu\mu)$, $a = \frac{\nu\mu}{2}$. Applying similar procedure of parallelization for lp processors as for transport equation, we write down the above 3-diagonal system in the form of $R \times R$ block system.

$$\begin{bmatrix} D & -A & \cdot & \cdot & \cdot \\ -A^T & D & -A & \cdot & \cdot \\ \cdot & -A^T & D & -A & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & -A^T & D \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \cdot \\ x_p \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \cdot \\ f_p \end{bmatrix},$$

where $p = lp - 1$. Coordinates f_0 and f_M contain additional terms coming from Dirichlet boundary conditions. Parallelization procedure [3] for this system is very similar to that described for transport equation. The fact that the block matrix is in the form $A = ae_R e_1^T$ enables to apply the above mentioned **"Divide and Conquer"** method. This procedure is a little more complicated than in the case of the transport equation. Now, **SCHUR system** is tree diagonal and of size $2(lp - 1)$. It is important that also in this case only one application of **MPI ALLREDUCE routine with option SUM** [5] gives resolution of the **SCHUR** system and hence very fast parallel resolution of the whole system.

3. **Nonlinear equations**
In our problem the nonlinear sub-problems are: nonlinear equations related to functions M_i corresponding to mixers

$$u_i{}' = \int_{\mathcal{A}} M_i(u_i, \cdot) d\beta,$$

and nonlinear equations of kinetics of chemical reactions

$$v_i{}' = K_i(v_i, \cdot).$$

It is proven in [2], that if the functions ρ_i are all bounded in the supremum norm, then the mixers

$$\int_{\mathcal{A}} M_i(\rho_i, \cdot)d\beta$$

satisfy Lipschitz conditions with respect to the arguments ρ_i. When the nonlinear part of the equation contains only mixer term, then we can solve approximately corresponding differential equations with help of implicit trapezoidal rule, which is A-stable. Hence, the simple iteration method applied to the resulting non-differential-nonlinear equation converges for τ small enough. This procedure was possible in the case of the model of turbulent flow without the chemical reactions terms (see [4] - also for results of numerical experiments).

Unfortunately, functions K_i are not all regular enough and hence corresponding nonlinear systems cannot be solved approximately with the above method, because iteration does not converge in a satisfactory way. In this situation we decided to apply for both nonlinear systems simple explicit Euler method:

$$u_{ik}^{n+1} = u_{ik}^{n} + \tau \int_{\mathcal{A}} M_i(u_i^n, \cdot)d\beta$$

with trapezoidal approximation of the integral over $\mathcal{A}$, and

$$v_{i,k}^{n+1} = v_k^n + \tau K_i(v_k^n, \cdot).$$

Since explicit method is not so much secure as implicit trapezoidal one, possibility of subdivision of original basic time-step τ up to chosen number of equal parts is provided for the whole nonlinear part of program.

6 Time Measuring on IBM Blue Gene/P "Notos"

"Notos" admits 4 processors per node, and in whole 1024 processors. Of course, communication is fastest inside one node. In order to enable time measurement using all resource of this system, we have chosen:

1. 4096 points on the interval [0,1.0] of x-axis,
2. $\mathcal{A} = [-40, 40]$ with 81 points,
3. T = 0.01,
4. $\nu = 0.01$, $\kappa = 0.1$,
5. 10 time-steps, hence basic time-step is $\tau = 0.001$,
   Moreover:
6. At each time-step, integral $\int_{\mathcal{A}} \rho(x, \alpha, \beta)d\beta$ was approximately computed with trapezoidal rule,
7. For nonlinear parts of algorithm (chemical reactions and mixers) basic time-step τ was divided by 10, hence nonlinear parts run ten times at each basic time step.

The computing time for one **complete** basic time-step without 'IO' was measured. Results of time measurements are given in Table 1. This table has to show how good is the "scaling" of this program. When "scaling" is optimal, computing time is inversely proportional to the number of processors used.

For these measurements

$$4,\ 8,\ 16,\ 32,\ 64,\ 128,\ 256,\ 512,\ 1024$$

processors were used. Hence, at each measurement step, the number of processors doubles.

In Table 1:

- column 1 gives the number of processors used;
- column 2 gives the computing time in seconds;
- column 3 gives the relative speedup $\boldsymbol{S_p}$, where:

$$\boldsymbol{S_p} = \frac{\textbf{time on one processor}}{\textbf{time on p processors}}\ .$$

Table 1. Measurement of computing time on "Notos"

1	2	3
4	110.16	4.000
8	55.10	7.997
16	27.67	15.924
32	14.06	31.339
64	7.27	60.610
128	3.49	126.257
256	2.21	199.385
512	1.62	272.000
1024	1.77	248.949

Remark. Observe that $\boldsymbol{c_{1024}}<\boldsymbol{c_{512}}$; this means that for certain p,

$$512 \le p \le 1024,$$

the speedup $\boldsymbol{S_p}$ begins to decrease. Clearly, the moment of this effect of saturation depends of sizes of all data used in this numerical experiment. The above mentioned saturation effect shows that in this case the application of more than 512 processors is rather useless.

7 Conclusion

Our numerical experiments on "Notos" IBM Blue Gene/P (4 processors per node), as well as on "Boreasz" IBM Power775 (32 processors per node), have shown that the applied method gives rather satisfactory results with respect to scalability of computation.

Let us observe that the size of the Schur matrix used in the linear part of our problem rises with the number of applied processors. In the situation when the size of the Schur matrix largely surpasses the size of a part of the main system matrix allocated to each processor, the "scaling" of the program begins to go down. This effect has appeared early when the number of grid-points on x-axis, and the number of points in the set $\boldsymbol{\mathcal{A}}$ of admissible velocities are relatively small. But in such a case the program works fast even when a small number of processors is used. Hence, the question is to find a satisfactory equilibrium between the numbers of grid-points on x-axis and in the set $\boldsymbol{\mathcal{A}}$ (accuracy!), and the expected speed of the program. This is very important to provide real-time simulations of the chemical process. Further numerical experiments will be needed to answer this question.

References

1. Janienko, N.N.: Metod drobnych šagov. Novosibirsk (perhaps there exists an English version) (1966).
2. Moszyński, K.: On certain numerical application of the time-splitting method. MIMUW IMSM report No. 201. http://www.mimuw.edu.pl (2011)
3. Bondeli, S.: Divide and conquer: a new parallel algorithm for solution of a tridiagonal linear system of equations. In: Burkhart, H. (ed.) CONPAR 90 - VAPP IV. LNCS, vol. 457, pp. 108–119. Springer, Heidelberg (1990)
4. Moszyński, K.: Simplified non Navier-Stokes model of turbulent flow and its first numerical realization in 2D. MIMUW IMSM report No. 203. http://www.mimuw.edu.pl (2011)
5. Barney, B.: Message passing interface. Lawrence Livermore National Laboratory. https://computing.llnl.gov/tutorials/mpi/
6. Barney, B.: OpenMP. Lawrence Livermore National Laboratory. https://computing.llnl.gov/tutorials/openMP/

Minisymposium on High Performance Computing Interval Methods

A Shaving Method for Interval Linear Systems of Equations

Milan Hladík(✉) and Jaroslav Horáček

Faculty of Mathematics and Physics, Department of Applied Mathematics, Charles University, Malostranské nám. 25, 118 00 Prague, Czech Republic
{hladik,horacek}@kam.mff.cuni.cz

Abstract. We propose an iterative improvement method for an enclosure of the solution set of a system of interval linear equations. The method sequentially cuts off (shaves) parts of a given enclosure that contain no solution, yielding thus tighter enclosures. Since shaving can be done independently in the coordinates, the procedure is easily parallelized. Our approach is convenient for problems with wide input intervals, where traditional methods give poor enclosures. Finally, we present a limited computational study.

Keywords: Interval systems · Interval matrix · Parallelization

1 Introduction

Solving systems of linear equations is a fundamental problem in linear and numerical algebra, and many other disciplines. Taking into account uncertain measurements, errors and other inexactness, and handling these uncertainties by the ranges of admissible values, we immediately face the problem of solving systems with interval coefficients. Considering all possible evaluations of interval data, the objective is to determine or at least tightly enclose all emerging solutions. It is known that computing the optimal bounds for the solution is an NP-hard problem [2,6] (even with a prescribed accuracy). That is why developing efficient algorithms for computing reasonably tight enclosures was a subject of intensive research. Nowadays, there are many methods published [2,4,7–9,14], but there is still enough open space for improvement.

In this paper, we propose a method, that, given an initial enclosure, iteratively cuts off some parts provably containing no solution. This approach is called *shaving*, which is borrowed from the area of constraint satisfaction problems [3,16].

Notation. The ith row of a matrix A is denoted by A_{i*}, the jth column by A_{*j}, and the unit vector by $e_i := (0,\dots,0,1,0,\dots,0)^T$. The vector $e = (1,\dots,1)^T$ is the vector of ones. An interval matrix is defined as

$$\boldsymbol{A} = [\underline{A}, \overline{A}] = \{A \in \mathbb{R}^{m \times n} \mid \underline{A} \leq A \leq \overline{A}\},$$

where $\underline{A} \leq \overline{A}$ are fixed matrices. The set of all $m \times n$ interval matrices is denoted by $\mathbb{IR}^{m \times n}$. By

$$A_c := \frac{1}{2}(\overline{A} + \underline{A}), \quad A_\Delta := \frac{1}{2}(\overline{A} - \underline{A})$$

we denote the midpoint and radius of $\boldsymbol{A}$, respectively. Interval arithmetic is defined, e.g., in books [7,8].

Interval Linear Systems. An interval linear system of equations is a family of systems

$$Ax = b, \quad A \in \boldsymbol{A},\ b \in \boldsymbol{b},$$

where $\boldsymbol{A} \in \mathbb{IR}^{n \times n}$ and $\boldsymbol{b} \in \mathbb{IR}^n$ are given. The corresponding solution set is defined as

$$\Sigma := \{x \in \mathbb{R}^n \mid \exists A \in \boldsymbol{A}\ \exists b \in \boldsymbol{b} : Ax = b\}.$$

Any interval vector $\boldsymbol{x} \in \mathbb{IR}^n$ containing Σ is called an enclosure of Σ. There are various methods for computing more or less tight enclosures of the solution set; see [2,4,7–9,14]. A linear relaxation method with iterative contraction of the enclosure was presented in [1]. Notice that a more general concept of solutions, so called AE-solutions, was studied, e.g., in [11,15].

2 A Shaving Method

Let $\boldsymbol{x}^0 \in \mathbb{IR}^n$ be an initial enclosure of the solution set Σ. The idea behind shaving methods is to take a slice $\boldsymbol{x}^0(\alpha, i)$ of $\boldsymbol{x}^0$ in the form of

$$\boldsymbol{x}^0(\alpha, i)_j = \begin{cases} \boldsymbol{x}^0_j & \text{if } j \neq i, \\ [\overline{x}^0_j - \alpha, \overline{x}^0_j] & \text{if } j = i, \end{cases} \tag{1}$$

where $\alpha \geq 0$ is a parameter. If we find that $\boldsymbol{x}^0(\alpha, i)$ contains no solution, then we cut off the slice and the tighter enclosure $\boldsymbol{x}^1$ reads

$$\boldsymbol{x}^1_j := \begin{cases} \boldsymbol{x}^0_j & \text{if } j \neq i, \\ [\underline{x}^0_j, \overline{x}^0_j - \alpha] & \text{if } j = i. \end{cases}$$

This procedure can be repeated for various $i \in \{1, \dots, n\}$, and similarly for shaving from below. Naturally, the larger α the more efficient the shaving.

To develop an efficient shaving method, we need to state some auxiliary results.

Lemma 1. *Let $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$ and $\boldsymbol{x} \in \mathbb{IR}^n$. Then the linear system*

$$Ax = b, \quad x \in \boldsymbol{x}$$

has no solution if and only if the linear system

$$A^T w + y - z = 0, \quad b^T w + \overline{x}^T y - \underline{x}^T z = -1, \quad y, z \geq 0 \tag{2}$$

is solvable.

Proof. By the well-known Farkas lemma (cf. [2]), the system $Ax = b$, $\underline{x} \leq x \leq \overline{x}$ has no solution if and only if the linear system

$$A^T w + y - z = 0, \quad b^T w + \overline{x}^T y - \underline{x}^T z < 0, \quad y, z \geq 0$$

is solvable. Now, (2) is obtained after normalization. $\square$

Now, we see that

$$Ax = b, \quad A \in \boldsymbol{A}, \ b \in \boldsymbol{b}, \ x \in \boldsymbol{x} \tag{3}$$

has no solution if and only if (2) is solvable for each $A \in \boldsymbol{A}$ and $b \in \boldsymbol{b}$. Checking this *strong solvability* is known to be computationally difficult (more precisely, co-NP-hard); see [2]. Below, we present an adaptation of the sufficient condition developed in [5].

A Sufficient Condition for Strong Solvability of (2). The sufficient condition consists of several steps. First, solve the linear programming problem

$$\min b_c^T w + \overline{x}^T y - \underline{x}^T z \ \text{ subject to } \ A_c^T w + y - z = 0, \ -e \leq w \leq e, \ y, z \geq 0,$$

and denote by w^*, y^*, z^* an optimal solution. The solution needn't be computed verified as it plays a role of a heuristic only. Suppose that the optimal value is negative. If it is not the case, then (2) is not solvable for $A := A_c$, $b := b_c$, and hence $\boldsymbol{x}$ contains a solution. If $y_i^* = 0$ for some i, then we fix the variable $y_i = 0$, and similarly for the entries of z. If the equation system

$$A^T w + y - z = 0, \quad b^T w + \overline{x}^T y - \underline{x}^T z = -1, \tag{4}$$

with $A \in \boldsymbol{A}$ and $b \in \boldsymbol{b}$, is square, we proceed along the lines given later below. If it is overdetermined, then it has the form of $A^T w = 0$, $b^T w = -1$. Since a positive multiple of w^* solves this system for $A := A_c$, $b := b_c$, we have that A_c is singular, which contradicts the assumption that Σ is bounded by $\boldsymbol{x}^0$. If (4) is underdetermined, then we put some additional equations to the system to be square. The left-hand side of the additional equations will be formed by an orthogonal basis of the null space of (4), and the right-hand side is calculated such that w^*, y^*, z^* solves the equations. Denote the resulting square interval system as

$$Cv = d, \quad C \in \boldsymbol{C}. \tag{5}$$

Let $v = (v^1, v^2)$, where v^1 consists of free variables appearing from w, and v^2 consists of non-negative ones originating from (y, z). Let $\boldsymbol{v}$ be an enclosure of its solution set (the tighter, the better). If $\underline{v}^2 \geq 0$, then (2) is solvable for each interval instantiation, implying that (3) is not strongly solvable.

Computing the Width of a Slice. Now, we employ the above ideas to handle the problem of determining as large as possible slice of $\boldsymbol{x}^0$ containing no solution. Since the slice $\boldsymbol{x}$ has the form of (1), it depends of the parameter $\alpha \geq 0$, and the interval system (5) depends on α, too. Thus, we have to determine a large value of α such that an enclosure to (5) satisfies the non-negativity condition $\underline{v}^2 \geq 0$.

The naive approach is to use a binary search for the optimal α. This would require solving a series of interval linear systems of equations. In the following, we give a simple method to calculating a feasible, not necessary optimal, value of α.

Due to (1), the system (5) depends on α in this way

$$(C + \alpha E_{ij})v = d, \quad C \in \boldsymbol{C}, \tag{6}$$

where $E_{ij} = e_i e_j^T$ is the matrix with 1 at position (i, j), and zeroes elsewhere.

Lemma 2. *Let v^* be a solution to $Cv = d$. Then the solution of $(C+\alpha E_{ij})v = d$ is $v^* - \frac{\alpha v_j^*}{1+\alpha C^{-1}_{ji}} C^{-1}_{*i}$.*

Proof. By the Sherman–Morrison formula for the inverse,

$$(C + \alpha E_{ij})^{-1} = (C + \alpha e_i e_j^T)^{-1} = C^{-1} - \frac{\alpha}{1 + \alpha C^{-1}_{ji}} C^{-1}_{*i} C^{-1}_{j*},$$

whence

$$(C + \alpha E_{ij})^{-1} d = C^{-1} d - \frac{\alpha}{1 + \alpha C^{-1}_{ji}} C^{-1}_{*i} C^{-1}_{j*} d = v^* - \frac{\alpha v_j^*}{1 + \alpha C^{-1}_{ji}} C^{-1}_{*i}.$$

□

Let $\boldsymbol{v}$ be an enclosure to the solution set of (5) (with $\alpha = 0$). By the above lemma, an enclosure to the solution set of (6) reads

$$\boldsymbol{v} - \frac{\alpha \boldsymbol{v}_j}{1 + \alpha \boldsymbol{C}^{-1}_{ji}} \boldsymbol{C}^{-1}_{*i}.$$

The denominator should be positive, otherwise it contains the zero. This gives the first restriction on α that

$$\alpha < -\frac{1}{\underline{\boldsymbol{C}^{-1}_{ji}}} \tag{7}$$

provided $\underline{\boldsymbol{C}^{-1}_{ji}} < 0$. As long as $\underline{\boldsymbol{C}^{-1}_{ji}} \geq 0$, there is no such an restriction on α. Next, denoting by I the index set of non-negative variables v^2, we get that

$$\boldsymbol{v}_k - \frac{\alpha \boldsymbol{v}_j}{1 + \alpha \boldsymbol{C}^{-1}_{ji}} \boldsymbol{C}^{-1}_{ki} \geq 0, \quad k \in I.$$

Since the left-hand side is an interval, its lower limit is required to be non-negative, i.e.

$$\underline{v}_k - \frac{\alpha}{1+\alpha \underline{\boldsymbol{C}_{ji}^{-1}}} \overline{\boldsymbol{v}_j \boldsymbol{C}_{ki}^{-1}} \geq 0, \quad k \in I.$$

Eliminating α, we obtain

$$\alpha \leq \frac{\underline{v}_k}{\overline{\boldsymbol{v}_j \boldsymbol{C}_{ki}^{-1}} - \underline{v}_k \underline{\boldsymbol{C}_{ji}^{-1}}}. \tag{8}$$

for each $k \in I$ such that $\overline{\boldsymbol{v}_j \boldsymbol{C}_{ki}^{-1}} > \underline{v}_k \underline{\boldsymbol{C}_{ji}^{-1}}$. From formulae (7) and (8) we determine the maximal α^* for eliminating the slice $\boldsymbol{x}$. In order that the result is reliable, the formulae should be evaluated by interval arithmetic (even though they contain real variables only).

The computational cost of this method for computing α^* is low. We have to calculate $\boldsymbol{v}$, an enclosure to (5), and $\boldsymbol{C}_{*i}^{-1}$, which is an enclosure to the solutions set of the interval system

$$Cu = e_i, \quad C \in \boldsymbol{C}.$$

In total, we need to solve only two interval linear systems of equations. On the other hand, the computed α^* may not be the largest possible width of the slice.

Remark 1 (Computational complexity). The computation of the width of a slide requires solving a linear program, finding an orthogonal basis of the null space and solving a sequence of interval linear systems. Thus, we arrive at the complexity of $\mathcal{O}(iter \cdot (LP + n^3))$, where LP is the running time for the linear program and $iter$ is the number of iterations. Therefore, the total computational time is

$$\mathcal{O}(iter \cdot n \cdot (LP + n^3)).$$

Since the number of iterations is typically very low, see Example 2, the overall cost is low, too.

Iterative Improvement. Since α^* needn't be optimal, we can think of improving it by repeating the whole process. We put $\alpha := \alpha^*$, and $\boldsymbol{v}$ will be an enclosure to (6). Similarly, $\boldsymbol{C}_{*i}^{-1}$ will be an enclosure to the solutions set of the interval system

$$(C + \alpha E_{ij})u = e_i, \quad C \in \boldsymbol{C}. \tag{9}$$

We determine the corresponding slice width α°, update $\alpha^* := \alpha^* + \alpha^\circ$ and repeat the process while improvement is significant (i.e., α° is large enough). Each iteration requires solving two interval systems, however, since the systems differ in one coefficient only, the new enclosures can be computed more effectively.

For instance, interval system solvers frequently use preconditioning by the (approximative) inverse of the midpoint matrix. Since the midpoint of (6) differs in the entry (i, j) only, its inverse is easily updated by using the Sherman–Morrison formula.

Updating the enclosure to (9) can be done even more efficiently. For a given $C \in \boldsymbol{C}$, we have by the Sherman–Morrison formula

$$(C + \alpha E_{ij})^{-1} = C^{-1} - \frac{\alpha}{1 + \alpha C^{-1}_{ji}} C^{-1}_{*i} C^{-1}_{j*}.$$

Its ith column draws

$$(C + \alpha E_{ij})^{-1}_{*i} = C^{-1}_{*i} - \frac{\alpha}{1 + \alpha C^{-1}_{ji}} C^{-1}_{*i} C^{-1}_{ji} = \frac{1}{1 + \alpha C^{-1}_{ji}} C^{-1}_{*i}.$$

Thus, $\boldsymbol{C}^{-1}_{*i}$ is updated as $\frac{1}{1+\alpha \boldsymbol{C}^{-1}_{ji}} \boldsymbol{C}^{-1}_{*i}$ without solving any system. Since the jth updated element $\boldsymbol{C}^{-1}_{ji}$ may be overestimated, we rather compute it by $\frac{1}{\alpha + 1/\boldsymbol{C}^{-1}_{ji}}$ instead of $\frac{1}{1+\alpha \boldsymbol{C}^{-1}_{ji}} \boldsymbol{C}^{-1}_{ji}$. In summary, while the first iteration needs to solve two interval systems, the others need to solve only one.

Remark 2. These results apply to parametric interval systems, too [11,12]. A parametric interval system is a system of the form

$$A(p)x = b(b), \quad p \in \boldsymbol{p},$$

where $A(p)$ and $b(p)$ depend on a vector of parameters p, whose domain is $\boldsymbol{p} \in \mathbb{IR}^k$. Thus the entries of the system vary within intervals, but not necessarily independently. Provided we employ a parametric interval solver [10,12], we immediately obtain a shaving method for parametric interval systems.

3 Examples and Numerical Experiments

Example 1. Consider the interval linear system $Ax = b$, where

$$A \in \boldsymbol{A} = \begin{pmatrix} -[6,7] & [8,10] \\ [5,6] & -[1,3] \end{pmatrix}, \quad b \in \boldsymbol{b} = \begin{pmatrix} -[10,11] \\ [-1,1] \end{pmatrix}.$$

The initial enclosure to the solution set Σ computed by the `verifylss` function from the package INTLAB [13] written under MATLAB framework is

$$\boldsymbol{x}^0 = ([-2.1891, 1.0385],\ [-3.2972, 0.1329])^T.$$

Let $i = 1$. The α-cut calculated by formulae (7)–(8) has the value of $\alpha_1 = 0.8521$. Iterative improvement makes no progress here, so we can reduce the upper limit of $\boldsymbol{x}^0_1$ by $\alpha_1 = 0.8521$.

Let $i = 2$. Now, the α-cut has width $\alpha_2 = 0.7142$. The iterative improvement process determines after other two cuts, $\alpha_3 = 0.1669$ and $\alpha_4 = 0.0657$. Thus, we can reduce $\boldsymbol{x}^0_2$ from above by the value of $\alpha_2 + \alpha_3 + \alpha_4 = 0.9468$.

Shaving from below fails since the initial enclosure is very tight from below. Therefore, we terminate with the resulting enclosure

$$\boldsymbol{x}^1 = ([-2.1891, 0.1864],\ [-3.2972, -0.8139])^T.$$

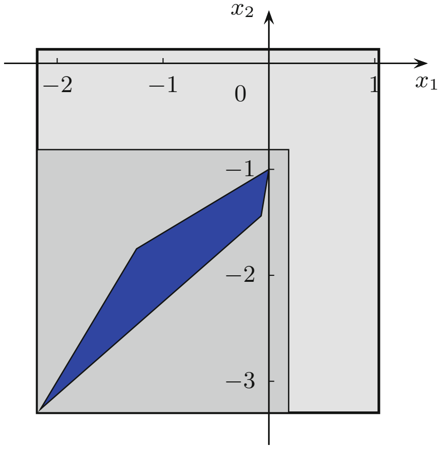


Fig. 1. (Example 1) The solution set is the blue (dark) polygon, the initial enclosure is in light gray, and the reduced enclosure in gray. (color figure online)

Notice that the interval hull of Σ is

$$\boldsymbol{x}^3 = ([-2.1579, 0], [-3.2632, -1])^T,$$

so the shaving method approaches the optimal enclosure. The solution set Σ, the initial enclosure $\boldsymbol{x}^0$, and the reduced enclosure $\boldsymbol{x}^1$ after the shaving are illustrated in Fig. 1.

Example 2. Herein, we present some randomly generated examples for various dimensions. The input data for the system $Ax = b$ were generated as follows. The entries of A_c and b_c were generated randomly in $[-10, 10]$ with uniform distribution. All radii of $\boldsymbol{A}$ are equal to the parameter $\delta > 0$.

The computations were carried out in MATLAB 7.11.0.584 (R2010b) on a six-processor machine `AMD Phenom(tm) II X6 1090T Processor`, CPU 800 MHz, with 15579 MB RAM. Interval arithmetics and some basic interval functions were provided by the interval toolbox INTLAB v6 [13].

Tables 1 and 2 display the results. Therein, n stands for the system size, δ for the radii of the intervals, `time` for the running time in seconds, and `cuts` denoted the total number of α-cuts applied. Each record is an average of 100 examples. The efficiency of the shaving is measured by `sum` and `prod` defined as

$$\texttt{sum} := \frac{\sum_{i=1}^{n} (x^1_\Delta)_i}{\sum_{i=1}^{n} (x^0_\Delta)_i}, \quad \texttt{prod} := \frac{\prod_{i=1}^{n} (x^1_\Delta)_i}{\prod_{i=1}^{n} (x^0_\Delta)_i},$$

where $\boldsymbol{x}^0$ is the initial interval and $\boldsymbol{x}^1$ the computed one. Thus, `sum` corresponds to the sum of interval widths of $\boldsymbol{x}^1$, while `prod` corresponds to the volume of $\boldsymbol{x}^1$, both related to $\boldsymbol{x}^0$.

Table 1. (Example 2) Randomly generated data without overall iterations.

n	δ	`time`	`sum`	`prod`	`cuts`
5	0.5	0.2568	0.7137	0.1355	13.02
10	0.25	0.6375	0.7522	0.04950	30.94
20	0.05	1.879	0.7848	0.02756	61.09
50	0.025	14.58	0.8569	0.03647	187.2
100	0.01	78.78	0.9049	0.04051	373.8

Table 2. (Example 2) Randomly generated data with overall iterations.

n	δ	`time`	`sum`	`prod`	`cuts`
5	0.5	0.4977	0.6465	0.07751	18.06
10	0.25	0.9941	0.6814	0.02184	45.06
20	0.05	3.136	0.7161	0.00639	87.77
50	0.025	26.65	0.8071	0.03424	281.9
100	0.01	228.5	0.8693	0.01531	946.3

Table 1 shows the results for the case, when the input interval vector is shaven in each coordinate just once from the above and once from the below (including iterative improvement). On the other hand, Table 2 shows the experiments for the same data, where the shaving was repeatedly applied until no significant cutting happen. We can observe that the overall iterations need approximately twice more time, but the resulting intervals are yet more tight.

The interval radius δ was chosen to be large to study behavior for "hard" instances. For narrow radii, the traditional methods yield tight enclosure, so the proposed shaving method is not suitable (and, indeed, makes little progress in reducing initial enclosures). Observe that with larger dimensions we decreased the parameter δ. This is because wider intervals would make the matrix singular (i.e., $\boldsymbol{A}$ would contain a singular matrix), and thus the solution set Σ would be unbounded. Therefore, when we increase n, we have to decrease δ.

4 Conclusion

We proposed a polynomial time iterative method for reducing the initial enclosure of the solution set of interval linear equations. The method cuts slices of the enclosure that certainly do not contain any solution. Numerical experiments discussed in the last section showed that the shaving method can effectively handle the hard instances with wide intervals and significantly reduce the initial overestimation.

Since we cut off independently in each coordinate from above and from below, this shaving procedure is suitable for parallelization. However, implementation of an efficient parallelization is not a straightforward task, and is left for the future research.

Acknowledgments. M. Hladík was supported by CE-ITI (GAP202/12/G061) of the Czech Science Foundation. J. Horáček was supported by the Czech Science Foundation Grant P402-13-10660S, and by the Charles University grant GAUK N0. 712912.

References

1. Beaumont, O.: Solving interval linear systems with linear programming techniques. Linear Algebra Appl. **281**(1–3), 293–309 (1998)
2. Fiedler, M., Nedoma, J., Ramík, J., Rohn, J., Zimmermann, K.: Linear Optimization Problems with Inexact Data. Springer, New York (2006)
3. Goldsztejn, A., Goualard, F.: Box consistency through adaptive shaving. In: Proceedings of the 2010 ACM Symposium on Applied Computing, SAC '10, pp. 2049–2054. ACM, New York (2010), http://doi.acm.org/10.1145/1774088.1774519
4. Hladík, M.: A new operator and method for solving interval linear equations (2013), http://arxiv.org/abs/1306.6739
5. Hladík, M.: Weak and strong solvability of interval linear systems of equations and inequalities. Linear Algebra Appl. **438**(11), 4156–4165 (2013)
6. Kreinovich, V., Lakeyev, A., Rohn, J., Kahl, P.: Computational Complexity and Feasibility of Data Processing and Interval Computations. Kluwer, Dordrecht (1998)
7. Moore, R.E., Kearfott, R.B., Cloud, M.J.: Introduction to Interval Analysis. SIAM, Philadelphia (2009)
8. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge University Press, Cambridge (1990)
9. Neumaier, A.: A simple derivation of the Hansen-Bliek-Rohn-Ning-Kearfott enclosure for linear interval equations. Reliab. Comput. **5**(2), 131–136 (1999)
10. Popova, E.D.: Webcomputing service framework. Int. J. Inf. Theor. Appl. **13**(3), 246–254 (2006)
11. Popova, E.D., Hladík, M.: Outer enclosures to the parametric AE solution set. Soft Comput. **17**(8), 1403–1414 (2013)
12. Popova, E.D., Krämer, W.: Inner and outer bounds for the solution set of parametric linear systems. J. Comput. Appl. Math. **199**(2), 310–316 (2007)
13. Rump, S.M.: INTLAB - INTerval LABoratory. In: Csendes, T. (ed.) Developments in Reliable Computing, pp. 77–104. Kluwer Academic Publishers, Dordrecht (1999). http://www.ti3.tu-harburg.de/rump/
14. Rump, S.M.: Verification methods: rigorous results using floating-point arithmetic. Acta Numer. **19**, 287–449 (2010)
15. Shary, S.P.: A new technique in systems analysis under interval uncertainty and ambiguity. Reliab. Comput. **8**(5), 321–418 (2002)
16. Trombettoni, G., Papegay, Y., Chabert, G., Pourtallier, O.: A box-consistency contractor based on extremal functions. In: Cohen, D. (ed.) CP 2010. LNCS, vol. 6308, pp. 491–498. Springer, Heidelberg (2010). http://dx.doi.org/10.1007/978-3-642-15396-9_39

Finding Enclosures for Linear Systems Using Interval Matrix Multiplication in CUDA

Alexander Dallmann(✉), Philip-Daniel Beck, and Jürgen Wolff von Gudenberg

Chair of Computer Science II, University of Würzburg, Am Hubland, 97074 Würzburg, Germany
alexander.dallmann@uni-wuerzburg.de

Abstract. In this paper we present CUDA kernels that compute an interval matrix product. Starting from a naive implementation we investigate possible speedups using commonly known techniques from standard matrix multiplication. We also evaluate the achieved speedup when our kernels are used to accelerate a variant of an existing algorithm that finds an enclosure for the solution of a linear system. Moreover the quality of our enclosure is discussed.

Keywords: GPGPU · Interval arithmetic · Linear algebra · Parallel computing

1 Introduction

Today graphics cards like the NVIDIA Tesla series are used in workstations as well as supercomputers to speed up computations using the highly parallel execution model of the GPU architecture. Especially linear algebra routines like matrix computations can be accelerated by outsourcing the computation to the GPU.

In this paper we present GPU routines to carry out interval matrix computations, as well as routines that perform a real matrix product with directed rounding. Also a routine for matrix computation in two-fold working precision is implemented using error-free transformations [1]. We then show how those routines can be applied to speed up a variant of an existing algorithm [2] for computing an enclosure for the solution of a linear system.

2 Related Work

In [2] the implementation of an algorithm that finds an enclosure for the solution of a linear system is described. The computation of a matrix-matrix product on CUDA in two-fold working precision is discussed in [1]. General optimizations for CUDA Kernels with matrix multiplication as an example are formulated in [3]. Reference [4] discusses a parallel variant of the Interval Newton Method on CUDA.

3 Preliminaries

3.1 Notation

Throughout the paper we will denote intervals by [], e.g. $[x]$. Interval vectors and matrices will be written with a bold letter, e.g. $[\boldsymbol{x}]$ or $[\boldsymbol{A}]$. For an indepth covering of interval arithmetic we refer to [5] and [6]

3.2 Computing a Verified Enclosure of a Linear System

Finding the solution of a real linear system $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$, with $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^n$, is a common numerical problem used in various applications. The result $\tilde{\boldsymbol{x}}$ of a numerical algorithm is usually some approximation of the real solution, having some unknown error term $\boldsymbol{e}$, so that $\boldsymbol{x} = \tilde{\boldsymbol{x}} + \boldsymbol{e}$. Using interval arithmetic an algorithm that finds verified enclosures of the solution $\hat{\boldsymbol{x}}$ can be implemented. The verified enclosure itself is obtained by applying Brouwer's fixed-point theorem.

The method we adapted is described in [2], where more details are given. It uses a Newton-like method to find an enclosure $[\boldsymbol{y}]$ for the residual of $\boldsymbol{Ax}$. To do so, the iteration scheme

$$\boldsymbol{y}^{\boldsymbol{k+1}} = \underbrace{\boldsymbol{Rd}}_{\boldsymbol{z}} + \underbrace{(\boldsymbol{I} - \boldsymbol{RA})}_{\boldsymbol{C}}\boldsymbol{y}^{\boldsymbol{k}}, k = 0, 1, ...$$

is used, with $\boldsymbol{R}$ being an approximate inverse of $\boldsymbol{A}$. By replacing all iterates with interval vectors, a result from [7] can be applied for this equation, that says that if $[\boldsymbol{y}]^{\boldsymbol{k}} \overset{\circ}{\subset} [\boldsymbol{y}]^{\boldsymbol{k+1}}$ holds for any index k, $\boldsymbol{R}$ and $\boldsymbol{A}$ are regular and there exists a unique solution $\boldsymbol{y} \in [\boldsymbol{y}]^{\boldsymbol{k}}$, where $\overset{\circ}{\subset}$ means contained in the interior.

Having found an approximate solution $\tilde{\boldsymbol{x}}$ for the original linear system, the enclosure of the residual can be used to give a verified enclosure of the solution by $\hat{\boldsymbol{x}} \in \tilde{\boldsymbol{x}} + [\boldsymbol{y}]^{\boldsymbol{k+1}}$

4 Implementation

All routines are implemented in C++/CUDA using version 5.0 of the CUDA SDK and use double-precision to carry out floating-point computations.

For interval computations directed rounding must be available. In CUDA the rounding-mode can be specified on an instruction level [8] using intrinsic functions [9]. Thus fine grained control over the rounding-mode is possible.

In order to speed up computation we use a tiled matrix multiplication as shown in Fig. 1. The result matrix is split into rectangular tiles and each tile is computed by its own thread block. Due to hardware limitations, in our case, the number of threads in a thread block is smaller than the number of cells in a tile. It follows that every thread needs to compute multiple cells of the tile.

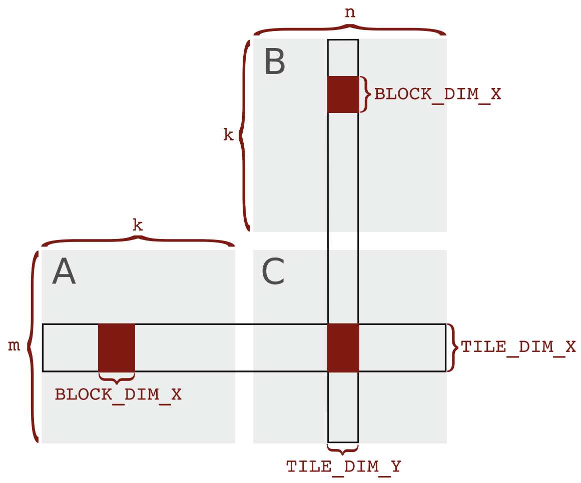


Fig. 1. Model for multiplication of two matrics $C^{m \times n} = A^{m \times k} \cdot B^{k \times n}$.

In [3] a scheme where every thread computes one or more rows of the tile is described. The shared memory is used to reduce global memory access while computing the rows. We adopted this approach for all our routines.

All routines have been tested with different tile and thread block sizes to determine the fastest combination. The tile and thread block sizes are varied between 256×16 and 64×8 and between 32×4 and 16×4 respectively.

We use the kernel template shown in Algorithm 1.1 for all our kernels. Only the computation of the scalar product is varied according to the specific case. Also kernels that don't use tiling were developed to demonstrate the achieved speed up.

Whenever appropriate the intrinsic FMA function [9] is used to speed up computation and avoid additional round-off errors. All kernels are implemented as C++ function templates and the decision for a concrete rounding-mode is made at compile-time to reduce runtime overhead.

4.1 Interval Matrix-Matrix Product

An interval matrix-matrix product was implemented using the tiling scheme described before. Since an interval consists of two floating-point numbers for the lower and upper bound, more shared memory and registers are used by the kernel compared to a kernel that executes floating-point matrix-matrix product. This results in smaller possible thread block sizes.

4.2 Real Matrix-Matrix Product with Directed Rounding

As of version 5.0, CUBLAS routines do not support directed rounding. We implemented a matrix-matrix multiplication routine that makes use of intrinsic

Algorithm 1.1. Basic kernel template for matrix-matrix multiplication that implements configuration in Fig. 1

```
Input: m, n, k, A, B, C
$a\_tile$, $b\_tile$, $c\_tile \leftarrow$ positions of tiles.
$results[TILE\_DIM\_Y] \leftarrow 0$ // initialize dot-product results with 0.
$shared\_cache[BLOCK\_DIM\_X][TILE\_DIM\_Y] \leftarrow 0$ // shared memory cache
$steps \leftarrow k/BLOCK\_DIM\_X$
$step \leftarrow 0$
while $step < steps$ do
    $shared\_cache \leftarrow$ load part of current sub-tile from b_tile
    $results\_pos \leftarrow 0$
    while $results\_pos < TILE\_DIM\_Y$ do
        // Compute next term of scalar product for every row element
        $results\_pos \leftarrow results\_pos + 1$
    end while
    $step \leftarrow step + 1$
end while
$c\_tile \leftarrow results$ // store row of result back
```

functions to support all in IEEE-754 [10] specified rounding modes. As mentioned before the decision for a specific rounding-mode is made at compile-time to ensure that no overhead occurrs. The routines were implemented using the same kernel template shown in Algorithm 1.1 to achieve a good performance.

4.3 Matrix-Vector Product as in Two-Fold Working Precision

A matrix-vector product in higher precision is needed by the algorithm implemented to demonstrate our routines. This is realized using error-free transformations to compute the dot product as in twice the working precision [11]. In [1] an implementation of a matrix-matrix product using error-free transformations is shown. We adapted this approach for our matrix-vector multiplication that evaluates the dot-product in twice the working precision.

5 Verification Algorithm

In Sect. 3.2, we presented the basics for implementing an iterative a posteriori method for calculating an enclosure of a solution $\hat{\boldsymbol{x}}$ of a linear system $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$. The start interval $[\boldsymbol{x}]^{\boldsymbol{0}}$ of an iterative a posteriori interval method does not neccessarily contain the correct solution $\hat{\boldsymbol{x}}$, but aims to find an enclosure after some iteration steps. In this section we describe some details of our implementation.

Algorithm 1.2 gives an overview of all the neccessary steps.

Algorithm 1.2. LinSolve($\boldsymbol{A}, \boldsymbol{b}, [\boldsymbol{x}]$) [2]

```
1. Calculation of an approximate solution
2. Real residual iteration to improve approximate solution
3. Computation of enclosures $[\boldsymbol{C}]$ and $[\boldsymbol{z}]$ for $\boldsymbol{C} = \boldsymbol{I} - \boldsymbol{RA}$ and $\boldsymbol{z} = \boldsymbol{R}(\boldsymbol{b} - \boldsymbol{A}\tilde{\boldsymbol{x}})$
4. Finding a verified enclosure of solution $\hat{\boldsymbol{x}}$
```

In step one, an approximate solution of $\tilde{\boldsymbol{x}}$ is calculated, using an approximation of the inverse matrix $\boldsymbol{R}$ of $\boldsymbol{A}$. We use existing routines from the MAGMA library for calculating the inverse. Therefore matrix $\boldsymbol{A}$ is LU-factorized by using Magama's *getrf* routine. With Magma's *getri* routine the inverse is subsequently determined and $\tilde{\boldsymbol{x}}^{(0)}$ is calculated as $\tilde{\boldsymbol{x}}^{(0)} = \boldsymbol{R} \cdot \boldsymbol{b}$.

After that the approximate solution $\tilde{\boldsymbol{x}}^{(0)}$ is refined using real residual iteration. In every iteration step the scalar products are evaluated in two-fold working precision to reduce rounding errors. The iteration is stopped after a fixed number of iterations or if the desired accuracy is reached.

The symbol $\boxminus$ is used to indicate that scalar products are evaluated in two-fold working precision.

In step three, the verification step is prepared by calculating enclosures of $[\boldsymbol{C}]$ and $[\boldsymbol{z}]$. This step can be seen in Algorithm 1.3. Symbol $\Diamond$ means, that an interval enclosure of the real result is calculated using directed rounding.

Algorithm 1.3. Computation of enclosures $[\boldsymbol{C}]$ and $[\boldsymbol{z}]$.

```
Input: $\boldsymbol{A}, \boldsymbol{R}, \tilde{\boldsymbol{x}}$
$[\boldsymbol{C}] \leftarrow \Diamond(\boldsymbol{I} - \boldsymbol{R} \cdot \boldsymbol{A})$;
$\boldsymbol{d} \leftarrow \boxminus(\boldsymbol{b} - \boldsymbol{A} \cdot \tilde{\boldsymbol{x}})$;
$[\boldsymbol{delta\_d}] \leftarrow \Diamond(\boldsymbol{b} - \boldsymbol{A} \cdot \tilde{\boldsymbol{x}} - \boldsymbol{d})$;
$[\boldsymbol{z}] \leftarrow \Diamond(\boldsymbol{R} \cdot \boldsymbol{d} + \boldsymbol{R} \cdot [\boldsymbol{delta\_d}])$;
return $[\boldsymbol{C}], [\boldsymbol{z}]$
```

During the verification step shown in Algorithm 1.4, the algorithm tries to calculate an enclosure for the real residual $\hat{\boldsymbol{y}}$ using an interval residual iteration. Since we are using an a posteriori method, the starting interval may not contain the searched fixed-point. The iteratees converge towards the fixed-point, but may not contain it. Using ϵ-inflation this problem can be reduced.

After an enclosure for the residual has been computed an interval containing the exact solution $\hat{\boldsymbol{x}}$ can be obtained $[\hat{\boldsymbol{x}}] = \tilde{\boldsymbol{x}} + [\hat{\boldsymbol{y}}]$.

6 Performance Measurements

Our performance tests were executed on a NVIDIA Tesla C2070 GPU with CUDA compute capability 2.0 and Fermi architecture. The host was running a Gentoo Linux 64 Bit system with an Intel Xeon E5504 quad-core CPU with 2 GHz and 8 GB RAM. NVidia Driver version 304.64 and CUDA SDK 5.0 were installed. For comparison with CXSC we used version 2.5.3.

Algorithm 1.4. VerificationStep [2]

```
Input: $[\hat{\boldsymbol{y}}], [\boldsymbol{z}], [\boldsymbol{C}]$
$\epsilon \leftarrow 1000; p_{max} \leftarrow 10; p \leftarrow 0; [\hat{\boldsymbol{y}}]^{(\boldsymbol{0})} \leftarrow [\boldsymbol{z}]$;
repeat
    $[\hat{\boldsymbol{y}}]^{(\boldsymbol{p})} \leftarrow [\hat{\boldsymbol{y}}]^{(\boldsymbol{p})} \cdot \epsilon$; {$\epsilon$-Inflation}
    $[\hat{\boldsymbol{y}}]^{(\boldsymbol{p+1})} \leftarrow \left([\boldsymbol{z}] + [\boldsymbol{C}] \cdot [\hat{\boldsymbol{y}}]^{(\boldsymbol{p})}\right)$;
    $IsVerified \leftarrow \left([\hat{\boldsymbol{y}}]^{(\boldsymbol{p+1})} \overset{\circ}{\subset} [\hat{\boldsymbol{y}}]^{(\boldsymbol{p})}\right)$;
    $p \leftarrow p + 1$;
until $IsVerified$ or $(p \geq p_{max})$
$[\hat{\boldsymbol{y}}] \leftarrow [\hat{\boldsymbol{y}}]^{(\boldsymbol{p})}$;
return $[\hat{\boldsymbol{y}}]$, $IsVerified$;
```

6.1 BLAS Routines

In Fig. 2 the performance of our fastest matrix-matrix multiplication kernel is compared to the current CUBLAS *dgemm* operation. As can be seen we reach a peak performance around 207 GFlops while CUBLAS peaks around 310 GFlops. Our kernel reaches roughly 66 % of the CUBLAS kernel performance so there is still room left for improvements. We assume that it should be possible to produce still faster versions of our interval routines.

When computing an interval matrix-matrix product every computation has to be carried out for the upper and lower bound. It follows that such a kernel needs more registers and shared memory as a corresponding floating-point kernel. When reaching the maximum usable registers a CUDA kernel spills over into global memory. Additional store and load instructions will be generated that

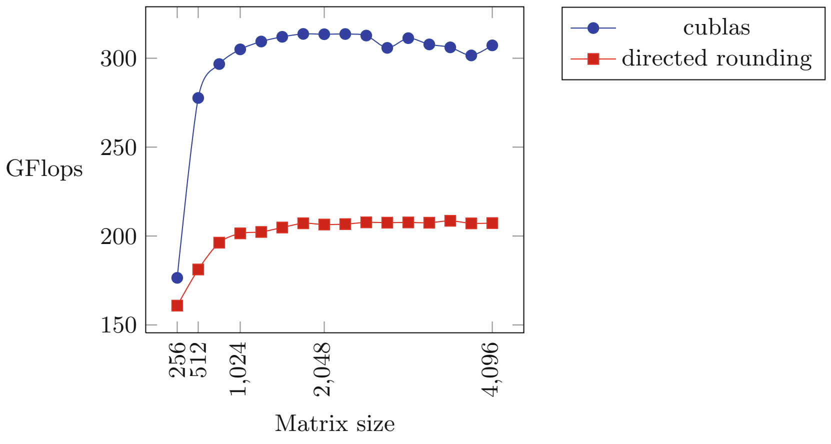


Fig. 2. Performance of our matrix-matrix multiplication implementation that supports directed rounding compared to current CUBLAS.

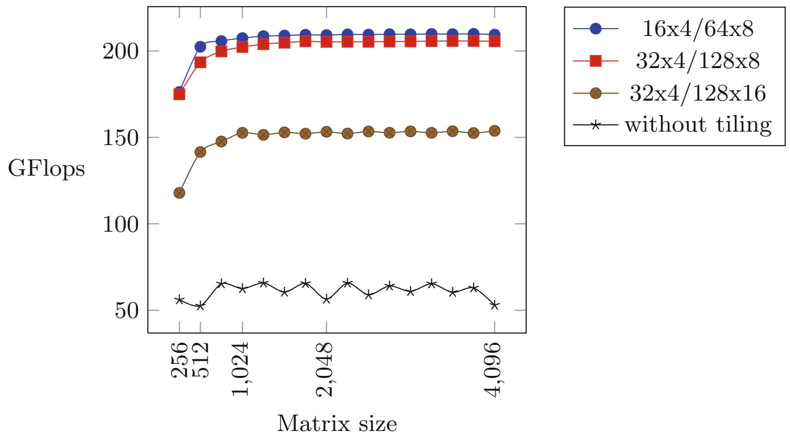


Fig. 3. Performance of interval matrix-matrix multiplication routines with different block and tile dimensions.

Table 1. A table showing resource allocation for kernels with different dimensions. Performance and Speedup against the routine without tiling are given for a problem size of 4096 × 4096. SM = Shared Memory; Regs = Registers; W/SMP = Warps/Shared Multiprocessor

No.	Block	Tile	Regs	SM	W/SMP	GFlops	Speedup
1	32×4	128×8	57	4096	16	205	3.94
2	32×4	128×16	63[a]	8192	16	153	2.94
3	16×4	64×8	61	2048	16	209	4.02
4	8×8	8×8	36	0	16	52	1

[a]Register limit is reached. Additional access to global memory is neccessary.

slow down the computation. In Fig. 3 we compare kernels that use different tile and thread-block sizes. Details about the kernels can be found in Table 1.

Although all kernels reach the same occupancy of 16 warps per multiprocessor kernel 2 is a lot slower because the maximum register limit is reached and additional store and load instructions to global memory are neccessary to correctly run the kernel. Kernel 1 and 3 are almost equally fast but since in kernel 3 the thread-block consists only of 64 threads it needs less shared memory, 8 blocks instead of 4 can be scheduled which seems to result in slightly better overall latency-hiding.

6.2 Verification Algorithm

For measurement of our verification algorithm implementation, we used routines from LAPACK to create random integer test-matrices. With these test-matrices we measured performance for our optimized CUDA implementation averaged over 10 test runs. For each run the measured time contains data transfer of

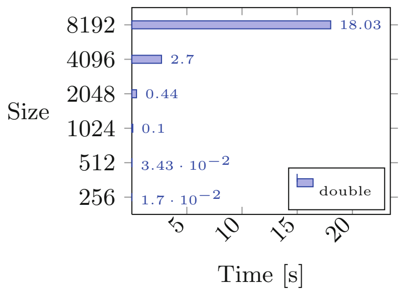


Fig. 4. Performance measurement results for the linear system solver using the optimized CUDA implementation

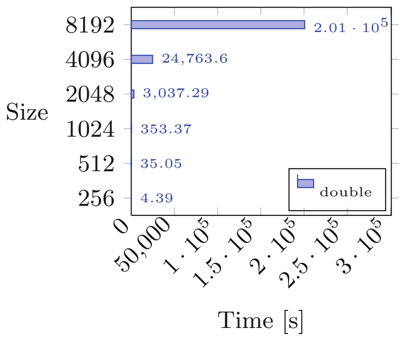


Fig. 5. Performance measurement results for CXSC - C++ verified toolbox implementation

matrices to the GPU, run time of the solving algorithm as well as copying back results from the GPU. Figures 4 and 5 shows time measurements of our implementation and the reference CXSC implementation for matrix sizes of 256 up to 8192. The maximum matrix size was limited by available memory on the GPU.

Besides the fact that our implementation uses the GPU, the main difference between our CUDA implementation and the compared CXSC implementation is the quality of scalar-product calculation. CXSC uses exact evaluation of dot products which results in tight enclosures of the exact floating-point dot product result. The drawback of this approach is that exact evaluation is computation intensive. In order to reduce accumulation of errors we use two-fold working precision for dot product calculations where appropriate, still rounding-errors accumulate and therefore, our enclosure is getting wider as matrix dimensions increase. For a random testcase this effect can be seen in Table 2. Since we use random integer test matrices CXSC finds the exact solutions, represented as point-interval vectors while the width of our results increases with the problem size. Overall, our implementation is less accurate, but as speedup also shows, much faster than CXSC.

Table 2. Interval vector width for optimized CUDA implementation and CXSC implementation for one test case

Size	Width(CUDA)	Width(CXSC)	Speedup
256	$7.92 \cdot 10^{-11}$	0	257
512	$2.9 \cdot 10^{-10}$	0	1,023
1,024	$7.13 \cdot 10^{-9}$	0	3,522
2,048	$1.39 \cdot 10^{-8}$	0	6,900
4,096	$4.53 \cdot 10^{-8}$	0	9,158
8,192	$1.89 \cdot 10^{-7}$	0	11,129

7 Conclusion

In this paper we developed interval matrix routines on CUDA and successfully applied them to an existing method for finding enclosures of solutions for linear systems. Applying common optimization techniques from floating-point matrix multiplication improved the performance of those routines. Implementing a known algorithm for finding the solution of a linear system showed that promising speedups can be achieved using the GPU. Since our routines suffer from loosing accuracy compared to exact but more computation intensive evaluation, investigations into exact evalution of scalar products on the GPU are planned in the future.

References

1. Fujimoto, N.: Economical two-fold working precision matrix multiplication on consumer-level CUDA GPUs. In: 2011 Second Workshop on Architecture and Multi-Core Applications (WAMCA), pp. 24–29 (2011)
2. Hammer, R.: C++ Toolbox for Verified Computing. Springer, Heidelberg (1995)
3. Cui, X., Chen, Y., Mei, H.: Improving Performance of Matrix Multiplication and FFT on GPU. In: 2009 15th International Conference on Parallel and Distributed Systems (ICPADS), pp. 42–48 (2009)
4. Beck, P.-D., Nehmeier, M.: Parallel interval newton method on CUDA. In: Manninen, P., Öster, P. (eds.) PARA. LNCS, vol. 7782, pp. 454–464. Springer, Heidelberg (2013)
5. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis. Springer, London (2001)
6. Alefeld, G., Herzberger, J.: Introduction to Interval Computations. Computer science and applied mathematics. Academic Press, New York (1983)
7. Rump, S.M.: Kleine Fehlerschranken bei Matrixproblemen. (Universität Karlsruhe 1980)
8. NVIDIA Corporation: Parallel Thread Execution ISA (Version 3.1). http://docs.nvidia.com/cuda/pdf/ptx_isa_3.1.pdf
9. NVIDIA Corporation: NVIDIA CUDA C Programming Guide (Version 5.0). http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
10. IEEE 754–2008: IEEE Standard for Floating-Point Arithmetic (2008)
11. Ogita, T., Rump, S., Oishi, S.: Accurate sum and dot product. SIAM J. Sci. Comput. **26**(6), 1955–1988 (2005)

GPU Acceleration of Metaheuristics Solving Large Scale Parametric Interval Algebraic Systems

Jerzy Duda(✉) and Iwona Skalna

AGH University of Science and Technology, Krakow, Poland
jduda@zarz.agh.edu.pl, skalna@agh.edu.pl

Abstract. A study on efficient solving of parametric interval linear systems using GPU computing is presented. Several illustrative examples from structural mechanics are employed to show that the proposed approach can significantly reduce computation time for this kind of problems. The stress is put on large uncertainties which are usually hard to be dealt with other, less time-consuming methods.

Keywords: GPU computing · Parametric interval linear systems · Metaheurisitcs · Truss structures

1 Introduction

The paper is devoted to the GPU acceleration of solving large-scale linear algebraic systems with elements that are nonlinear functions of parameters varying within prescribed intervals. The problem of solving such systems is of a great importance for reliability and risk analysis of civil engineering systems, among others. The goal of the *realised* numerical computations is to obtain a so-called hull solution, i.e., the tightest interval vector that encloses the solution set of parametric interval linear systems. In general, the problem of computing the hull solution is NP-hard, so the classical numerical approach tends to be computationally extensive and ineffective. Better approximations can be usually obtained using metaheuristic methods, which are in general designed to solve complex optimisation problems. Nevertheless, for larger instances of the problem the computational time is unacceptable. In this paper, the GPU acceleration techniques are used to speed up metaheuristic strategies. The proposed approach is presented in Sect. 3. The rest of the paper is organised as follows. Parametric interval linear systems are described in Sect. 2. In Sect. 4, computational experiments for illustrative problem instances from structural mechanics with different number of intervals as well as different uncertainty ranges are presented in order to verify the usefulness of the proposed approach. The results for accelerated and non-accelerated metaheuristics based on evolutionary algorithm and differential evolution are presented. The paper ends with directions for future study and concluding remarks.

2 Parametric Interval Linear Systems

A parametric linear system

$$A(p)x(p) = b(p), \tag{1}$$

is a system of linear equations with coefficients that are, in general, nonlinear functions of model parameters $(p_1, \ldots, p_K)$:

$$\begin{aligned} a_{ij}(p) &= a_{ij}(p_1, \ldots, p_K), \\ b_i(p) &= b_i(p_1, \ldots, p_K), \quad i, j = 1, \ldots, n. \end{aligned} \tag{2}$$

Very often, due to the scarcity or lack of data, the parameters p_k are unknown. This kind of uncertainty can be conveniently modelled using the interval approach ([1,4,5]), where an uncertain parameter p_k is bounded by a closed interval $\boldsymbol{p}_k = [\check{p}_k - \Delta p_k, \check{p}_k + \Delta p_k]$. The centre $\check{p}_k$ of $\boldsymbol{p}_k$ is considered as an approximation of p_k and $\Delta p_k > 0$ is an upper bound for the error of that approximation. Obviously, appropriate interval methods are required to correctly propagate interval uncertainty through a calculation (see, e.g., [1,4,5]).

Now, if the parameters are assumed to range within prescribed intervals, $p_k \in \boldsymbol{p}_k$ $(k = 1, \ldots, K)$, the family of parametric linear system is obtained

$$A(p)x(p) = b(p),\ p \in \boldsymbol{p}. \tag{3}$$

This family is called a *parametric interval linear system (PILS)* in order to underline its direct connection with interval linear systems and is usually written in the following compact form

$$A(\boldsymbol{p})x(\boldsymbol{p}) = b(\boldsymbol{p}). \tag{4}$$

The set of all solutions to the point linear systems from the family (3)

$$S(\boldsymbol{p}) = \{x \in \mathbb{R}^n \mid \exists\, p \in \boldsymbol{p}\ \ A(p)x = b(p)\} \tag{5}$$

is called a *parametric solution set*. It is generally of a complicated non-convex structure [2]. In practise, therefore, an interval vector $\boldsymbol{x}^*$, called an *outer interval solution*, satisfying $S(\boldsymbol{p}) \subseteq \boldsymbol{x}^*$ is computed instead. The tightest outer solution, with respect to the inclusion property, is called a *hull solution* or simply a *hull* and is defined by the following formula:

$$\square S(\boldsymbol{p}) = \bigcap \left\{ \boldsymbol{Y} \in \mathbb{IR}^n \mid S(\boldsymbol{p}) \subseteq \boldsymbol{Y} \right\}. \tag{6}$$

It is quite obvious that the problem of computing the hull can be formulated as a problem of solving the family of the following $2n$ constrained optimisation problems:

$$\begin{aligned} \underline{x}_i &= \min\{x(p)_i \mid A(p)x(p) = b(p),\, p \in \boldsymbol{p}\}, \\ \overline{x}_i &= \max\{x(p)_i \mid A(p)x(p) = b(p),\, p \in \boldsymbol{p}\}, \quad i = 1, \ldots, n. \end{aligned} \tag{7}$$

Theorem 1. *Let $A(\boldsymbol{p})x(\boldsymbol{p}) = b(\boldsymbol{p})$ and let $\underline{x}_i$ and $\overline{x}_i$ denote, respectively, the solution of the i-th minimisation and maximisation problem (7). Then, the hull solution*

$$\square S(\boldsymbol{p}) = \square\{x(p) : A(p)x(p) = b(p), p \in \boldsymbol{p}\} = [\underline{x}_1, \overline{x}_1] \times ... \times [\underline{x}_n, \overline{x}_n]. \tag{8}$$

Solving the problem (7) is a complex task, which stems mainly from the fact that the function to be optimised is given implicitly. Obviously, the complexity also grows with the size of the problem. Some experiments with using a global interval optimisation (IGO) approach and metaheuristics strategies to solve the problem (7) were already made. Each of those two competing approaches has pros and cons. Interval global optimisation guaranties reliable results, i.e., $\square S(\boldsymbol{p}) \subseteq \boldsymbol{x}^*_{IGO}$, where $\boldsymbol{x}^*_{IGO}$ denotes the result of IGO, but for larger problem and larger uncertainties, the method is very inefficient. Metaheuristic strategies seem to be winning when large uncertainties are involved. Nevertheless, for larger problems the computational time is unacceptable. In this survey, an attempt to decrease the computational time is made. To achieve this, the GPU-based metaheuristic strategies are developed.

3 Solving Systems of Linear Equations with GPU Acceleration

GPU [8] computing offers unprecedented application performance by offloading compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU. From a user's perspective, applications that requires extensive calculations simply run significantly faster. Although the performance of the mainstream GPU devices for double precision is limited comparing to the dedicated GPGPU devices like NVIDIA Testla, they can still offer a huge speed improvement for intensive calculations over traditional CPU units.

Contemporary GPUs and their programming environments have evolved to the point where many real-world applications can be easily implemented on them and run significantly faster than on multi-core systems. Today's modern computing architectures are built as hybrid systems with parallel-core GPUs working in tandem with multi-core CPUs.

CPU+GPU is a powerful combination, because CPUs consist of a few cores optimised for serial processing, while GPUs consist of thousands of smaller, more efficient cores designed for parallel processing of the same operations on different data. Thus the algorithms should be developed in such a way that serial portions of the code are run on the CPU while parallel portions are run on the GPU.

The development of GPU accelerated applications can be further facilitated by the use of popular numerical linear algebra libraries like BLAS (Basic Linear Algebra Subroutine) [9] and LAPACK (Linear Algebra PACKage) [10]. The most popular adaptations of these libraries to GPGPU environment are cuBLAS[11] from NVIDIA (only BLAS procedures), CULA [12] provided by EM Photonics (free for single type procedures) and MAGMA [13] developed at the University of Tennessee (open source license). For the purpose of the numerical experiments presented in the following sections, procedures implemented in the CULA library have been used, as the library can be easily integrated with Visual C++. However, similar results (at least in terms of magnitude) should be obtained with the use of other GPU accelerated LAPACK libraries.

3.1 GPU Acceleration Strategy

The parametric interval linear systems considered in the paper (for trusses with more than 100 nodes) are hard to be solved using standard approach. In our previous studies we have shown that metaheuristics like genetic algorithms [7] or differential evolution [3] can provide a good approximation of hull solution, however, the computation time for real large problems is significant. In this study we will then focus on the acceleration of metaheuristics solving large size PILS.

Most of the computation time in both metaheuristics is devoted to fitness evaluation of solutions - according to C++ profiling tools it takes 88–99 % of total execution time. This procedure involves calculation of a stiffness matrix (75 % of the fitness calculation time) and solving the system of linear equations (25 %). If BLAS library is used to generate the stiffness matrix, the second procedure becomes the most time consuming (72 % of total fitness calculation). Thus the most profitable is to transfer both calculation of the stiffness matrix (BLAS supported or not) and solving of the equations system from CPU to GPU.

3.2 GPU Acceleration of Stiffness Matrix Calculation

In order to asses whether to use GPU acceleration of the stiffness matrix calculation with the application of BLAS library (more specifically its DGEMM procedure) three variants of GPU accelerations have been tested. In the first variant both DGEMM and DGESV procedures were accelerated by GPU using CULA library, while in the second only equations solving is accelerated by GPU (DGESV procedure from CULA library), but the stiffness matrix is computed by CPU (DGEMM procedure from OpenBLAS library). Finally in the third variant, we wrote a dedicated CUDA kernel for the stiffness matrix calculation and used CULA library for the GPU acceleration of equations solving. The code of the kernel has been shown in in Fig. 1.

The results of the experiments with evolutionary algorithm (EA) with all three variants of GPU acceleration of DGEMM and DGESV procedures, run for the three different problems described in details in the next section, have been gathered in Table 1. Last column contains execution times of EA with DGEMM and DGESV procedures performed on CPU only.

The tests have shown that GPU acceleration of DGEMM procedure from CuBLAS library did not contributed to the decrease of computation time for the stiffness matrix, but caused a significant increase of this time (over 80 % for the largest problem). Development of a dedicated CUDA kernel for the calculation of the stiffness matrix also did not provide any spectacular improvement - for the largest problem this variant achieved 6 % advantage over

```
$i \longleftarrow blockDim.x * blockIdx.x + threadIdx.x$
$j \longleftarrow blockDim.y * blockIdx.y + threadIdx.y$
if $i < N$ AND $j < N$ then
  $i \longleftarrow 0$
  for $k \longleftarrow 0$ to $NE$ do
    if $A_{k*N+i} \neq 0$ AND $A_{k*N+j} \neq 0$ then
      $s \longleftarrow s + A_{k*N+i} * A_{k*N+j} * B_k$
    end if
    $C_{i*N+j} \longleftarrow s$
  end for
end if
```

Fig. 1. Outline of CUDA kernel for stiffness matrix calculation

Table 1. Times [s] for GPU accelerated fitness calculation (stiffness matrix calculation/equation solving)

Problem	Both CULA	CULA/CPU	Kernel/CULA	Both CPU
*Example*1	1.94	0.72	2.19	2.43
*Example*2	125.49	70.55	77.36	73.22
*Example*3	1364.17	749.61	707.11	1907.84

non-GPU accelerated DGEMM procedure. The lack of improvement that has been observed for the GPU acceleration of the stiffness calculation could be caused by insufficient transfer rates of data from host to GPGPU device and in the opposite direction (average transfer bandwidth in the test environment was 1.5 GB/s) comparing to the computing power of the GPGPU device itself (NVIDIA GTX 650Ti graphics card). DGEMM procedures had two be called twice as the calculation of the stiffness matrix required two operations of matrix multiplication. For the largest problem, however, all variants of GPU acceleration performed faster when compared to CPU (variant with CUDA kernel was 2.7 times faster).

3.3 GPU Acceleration of Linear Equations Solving

A standard LAPACK library contains two procedures for solving linear equations DGESV and DSGESV. Both of them use LU decomposition with partial pivoting. LU decomposition (also known as LU factorisation) factors a matrix as the product of a lower triangular matrix and an upper triangular matrix. Once an LU decomposition is obtained, the system $Ax = b$ can be solved. Gaussian elimination inverts L, so that from $Ax = LUx = b$, the system $Ux = L^{-1}b$ is obtained. Then backward substitution finds x. Gaussian elimination uses partial pivoting. The row with the biggest p_i in absolute value is brought into the pivoting position, i.e., if $|p_i| = \max_{j=1,2,\ldots,m} |p_j|$, then the row of p_1 is swapped with the row of p_i and p_i is used as a pivot. This simple idea results in a dramatic improvement in

Table 2. Times [s] for equation solved by GPU accelerated DGESV and DSGESV procedures

n	DGESV	DSGESV	Ratio
16	1.20	3.91	3.26
189	74.64	150.56	2.02
434	717.77	1144.70	1.59
560	1590.94	2296.01	1.44

the stability of the Gaussian elimination citation. DSGESV procedure first tries to decompose the matrix using single precision and uses this factorisation within an iterative refinement procedure to obtain a solution with double precision. Using iterative refinement maybe profitable in terms of solving time in GPGPU environment, as single precision operations are usually performed much faster than their double precision equivalents. We performed experiments whether to use GPU accelerated DGESV or DSGESV procedure (from CULA library) in the metaheuristics. Times obtained for the both procedures that solved 32*n equations used in the fitness evaluation procedure for the metaheuristics solving parametric interval linear systems are provided in Table 2.

Base on the above experiments DGESV procedure (without iterative refinement) was used in the main experiments, as for the analysed problems it performed faster than DSGESV procedure (with refinement). However, this trend decreased along with the problem size, and for very large problems DSGESV procedure could be more profitable than DGESV.

4 Numerical Experiments

The effects of GPU acceleration for the metaheuristics have been tested on the basis of three exemplary truss structures, each of different size: one floor four bay two floor truss, five bay six floor truss and ten bay eleven floor truss. All experiments were performed on a desktop computer with Intel E8300 CPU, 4 GB DDR2 RAM and NVIDIA GTX 650Ti GPU with 1 GB GDDR5 RAM. The algorithms were coded in Microsoft VC++ 2008 with CUDA 5.01 platform and CULA Dense R16a library installed.

Example 1. A small one floor four-bay truss (see Fig. 2) is first considered. There are 10 nodes and 21 elements (bars). The elements are joint at nodes using flexible, rotary joints. The truss is fully supported at nodes 1 and 5 which means that there are 16 degrees of freedom. The elements of the truss have the cross-section area $A = 0.0001\,\mathrm{m}^2$ and modulus of elasticity is uncertain $E = 200\mathrm{GPa}\pm 10\,\%$. Also the load acting downward is uncertain $F = 20\mathrm{kN}\pm 20\,\%$.

Displacements of two selected nodes, node 3 and node 8, are given in the Table 3. Differential evolution gave a little better results (wider interval) than evolutionary method. The execution time of 100 generations and 30 individuals for both algorithms was similar.

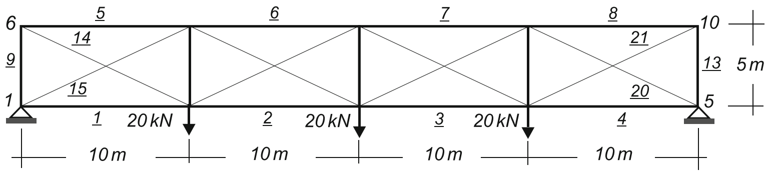


Fig. 2. 4-bay 1-floor truss

Table 3. Displacement of selected nodes for *Example* 1

Method	Node 3	Node 8	Time [s]
EOM	[-0.0058263707, 0.0058538249]	[-0.0034941384, -0.0013484597]	106.81
DE	[-0.0058699821, 0.0058706921]	[-0.0034963978, -0.0013457917]	109.55

Example 2. As a second example a finite element model for a 9-bay 9-floor truss is considered. There are 100 nodes and 342 elements, resulting in 189 variables and 342 uncertain parameters. The cross-section area $A = 0.005\,\text{m}^2$ and Young's modulus is uncertain $E = 200\text{GPa}\pm 10\,\%$. The uncertain load $F = 10\text{kN}\pm 20\,\%$ (Figs. 3).

The horizontal and vertical displacements of the upper left node are given in Table 4. This time 40 generations and 16 individual were computed for each algorithm. The computational time was significant (over 20 min), and was comparable with the time achieved by the metaheuristics without GPU acceleration, when BLAS library was applied for the calculation of stiffness matrix (version without BLAS run 3 times slower). According to our experiments, if the accuracy could be sacrificed, GPU procedures using float numbers instead of double would save additional 52 % of time.

Example 3. The third and last example considers a finite element model for a 14-bay 14-floor truss. There are 225 nodes and 812 elements, resulting in 434

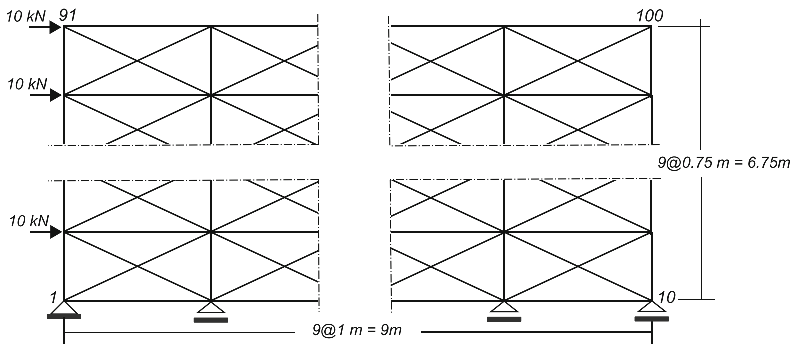


Fig. 3. 9-bay 9-floor truss

Table 4. Displacement of selected nodes for *Example* 2

Method	Node 91	Node 100	time [s]
EOM	[0.0000325971, 0.0000442695]	[0.0001311467, 0.0001702184]	1482.81
DE	[0.0000281077, 0.0000512565]	[0.0001097428, 0.0001879121]	1474.20

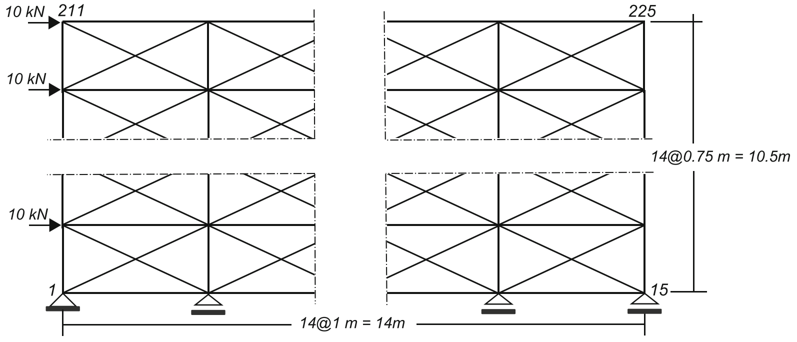


Fig. 4. 14-bay 14-floor truss

Table 5. Displacement of selected nodes for *Example* 3

Method	Node 211	Node 225	time [s]
EOM	[0.0002260031, 0.0002810077]	[0.0002781862, 0.0003191969]	14081
DE	[0.0001999374, 0.0003216216]	[0.0002361739, 0.0003721374]	14079

variables and 812 uncertain parameters. The cross-section area $A = 0.005\,\mathrm{m}^2$ and Young's modulus is uncertain $E = 200\text{GPa}{\pm}15\,\%$. The uncertain load $F = 10\text{kN}{\pm}15\,\%$ (Figs. 4).

The horizontal and vertical displacements of the most upper left node are given in Table 5. Like in the previous cases differential evolution achieved more accurate solutions than evolutionary method. The calculations were performed for 40 generations and 16 individuals and the computation time even with GPU acceleration was very long (almost 4 hours). However, it was almost 3 times shorter when no GPU acceleration was used. Additional 26 % of time would be saved if single precision accuracy could be sufficient.

5 Conclusions

The paper presents the study on the acceleration of selected population-base metaheuristics for solving parametric interval linear systems. Both evolutionary algorithm and differential evolution approach can be used to achieve good approximation of the hull solution for such systems. However, for the problems of a very large size, the computation time for metaheuristics become enormous. As the most time consuming operation is the fitness evaluation of solutions in a population, this task is the best candidate to be moved from CPU to GPU. Continuous development of GPU technology allows to achieve significant acceleration of algebraic calculations for a relatively small cost, even if double precision is required. Classic procedures that can be found in BLAS and LAPACK libraries have been ported into GPU accelerated libraries like CULA or MAGMA. However, the application of such libraries should be well thought and tested, as the benefits of GPGPU computing may not always outweigh the costs of data transfers to and from CPU, especially in budget systems. Writing a dedicated kernel, limiting the transfers, can be one of the solutions to this problem. Nevertheless further improvement in GPU technology will undoubtedly allow for greater reduction of computation time allowing metaheuristics to achieve good hull solution approximation for large PILS in an acceptable time.

References

1. Alefeld, G., Herzberger, J.: Introduction to Interval Computations (transl. by J. Rokne from the original German ‘Einführung In Die Intervallrechnung’), pp. xviii–333. Academic Press Inc., New York (1983)
2. Alefeld, G., Kreinovich, V., Mayer, G.: The shape of the solution set for systems of interval linear equations with dependent coefficients. Mathematische Nachrichten **192**(1), 23–36 (2006)
3. Duda, J., Skalna, I.: Differential evolution applied to large scale parametric interval linear systems. In: Lirkov, I., Margenov, S., Waśniewski, J. (eds.) LSSC 2011. LNCS, vol. 7116, pp. 206–213. Springer, Heidelberg (2012)
4. Moore, R.E.: Interval Analysis. Prentice-Hall Inc., New York (1966)
5. Neumaier, A.: Interval Methods for Systems of Equations, pp. xvi–255. Cambridge University Press, Cambridge (1990)
6. Rohn, J., Kreinovich, V.: Computing exact componentwise bounds on solutions of linear systems with interval data is np-hard. SIAM J. Matrix Anal. Appl. (SIMAX) **16**, 415–420 (1995)
7. Skalna, I., Duda, J.: A comparison of metaheurisitics for the problem of solving parametric interval linear systems. In: Dimov, I., Dimova, S., Kolkovska, N. (eds.) NMA 2010. LNCS, vol. 6046, pp. 305–312. Springer, Heidelberg (2011)
8. GPU accelerated computing. http://www.nvidia.com/object/what-is-gpu-computing.html Accessed 15th May2013
9. Lawson, C.L., Hanson, R.J., Kincaid, D., Krogh, F.T.: Basic linear algebra subprograms for fortran usage. ACM Trans. Math. Softw. **5**, 308–323 (1979)
10. LAPACK: Linear Algebra PACKage. http://www.netlib.org/lapack Accessed 15th May 2013
11. CUDA CUBLAS Library. PG-05326-032_V02. NVIDIA Corporation http://docs.nvidia.com/cuda/pdf/CUDA_CUBLAS_Users_Guide.pdf (2010). Accessed15th May 2013
12. CULAtools: GPU accelerated linear algebra. http://www.culatools.com Accessed 15th May 2013
13. Matrix algebra on GPU and multicore architectures. http://icl.cs.utk.edu/magma Accessed 15th May 2013

Parallel Approach to Monte Carlo Simulation for Option Price Sensitivities Using the Adjoint and Interval Analysis

Grzegorz Kozikowski[1] and Bartłomiej Jacek Kubica[2](✉)

[1] School of Computer Science, University of Manchester, Manchester, UK
grzegorz.kozikowski@mbs.ac.uk
[2] Institute of Control and Computation Engineering,
Warsaw University of Technology, Warsaw, Poland
bkubica@elka.pw.edu.pl

Abstract. This paper concerns a new approach to evaluation of Option Price sensitivities using the Monte Carlo simulation, based on the parallel GPU architecture and Automatic Differentiation methods. In order to study rounding errors, the interval arithmetic is used. Considerations are based on two implementations of the algorithm – the sequential and parallel ones. For efficient differentiation, the Adjoint method is employed. Computational experiments include analysis of performance, uncertainty error and rounding error and consider Black-Scholes and Heston models.

Keywords: Option pricing · The greeks · Automatic Differentiation · The Adjoint · Calibration · Interval analysis · CUDA

1 Introduction

Nowadays, banks, investment funds and other asset management companies demand accurate and fast market forecasts to make profitable and more mature investments. In the recent years, one of major instruments traded by these institutions is a financial option. The options are now traded on many exchanges throughout the world and take different commodities into account.

In general, the option is a contract that gives the holder the right to make further investments whose conditions are well-known at present. There exist two fundamental, different types of options. First, a call option gives a holder the opportunity to buy the underlying asset by a specific date for a certain price. When, the second - a put option is a contract that allows to sell the underlying asset at a fixed time for a given price. The financial value of these contracts can have positive or negative aspects and it is dependent on further movements in the price of the asset until the expiration date.

For this reason, option pricing and evaluation of option price sensitivities play an important role in hedging and risk management of assets. This paper proposes an approach using GPGPU technology and Automatic Differentiation to compute them efficiently. (see, e.g., [7,13]).

2 Option Pricing

2.1 General Approach to Pricing Options

In general, there are several factors affecting further price of a stock option:

- current stock price and strike price – as an option is exercised at some time in the future, the pay-off for a call option is a difference between stock price and strike price. Considering put options, the pay-off will be amount by which the strike price exceeds the stock price [7].
- time to expiration – for some specific options (American, exotic, etc.) a pay-off of an option increases with the time to expiration [7].
- volatility of stock price – is a measure of how unpredictable are future stock price movements [7].
- risk-free interest rate – as interest rates in the economy increase, the expected growth rate of the stock price tends to increase [7].
- dividends – have the effect of reducing the stock price on the ex-dividend date [7].

Most of existing financial models for option market take into account the factors above. Each model of the behaviour of the stock price is usually based on stochastic calculus and considered as a stochastic process (changes of the value are uncertain over time and are dependent on probability). Besides the stock price, this process predicts the volatility and it is called the stochastic volatility model, as well. In this model, the further price of asset S satisfies the following differential equation [7]:

$$dS(t) = \alpha S(t)dt + \sigma(t)S(t)dW(t) \tag{1}$$

where $\sigma(t)$ denotes a volatility and $W(t)$ is a geometric Brownian motion. The volatility is often represented as a function of several input-variables (except the Black-Scholes model):

$$\sigma(t) = f(Y(t)), \tag{2}$$

$$dY(t) = (a + bY(t))dt + cY(t)dV(t), \tag{3}$$

where a and b are constants and $dV(t) = \rho dW(t) + \sqrt{1-\rho^2}dZ(t)$. The processes V and W are correlated.

Black-Scholes Model. It assumes that stock price follows a generalized Wiener process with a constant expected drift rate and a constant variance rate. For the Black Scholes model, the asset price satisfies the following stochastic differential equation:

$$dS(t) = \alpha S(t)dt + \sigma(t)S(t)dW(t) \ , \tag{4}$$

where $W(t)$ is a generalized Wiener process, $\sigma(t)$ is the volatility of stock price and α is its expected rate of return.

Heston Model. This model was proposed by Steven Heston in 1993. Heston introduced an intuitive extension of the Black Scholes model and he assumed that the stock price follows the following diffusion process [12]:

$$dS_{t+1} = \mu S_t dt + \sqrt{\sigma_t} S_t \cdot W_t^1 \ . \tag{5}$$

For the Heston Model, the volatility is represented as a mean reverting stochastic process of the form:

$$d\sigma_{t+1} = \kappa \cdot (\theta - \sigma_t) \cdot dt + \xi \cdot \sqrt{\sigma_t} dW_t^2 \ . \tag{6}$$

where: θ is the long-term variance, κ – the mean reversion of volatility, ξ – the volatility of volatility and σ_0 – the initial volatility. dW_t^1 and dW_t^2 are correlated random variables with normal distribution (the correlation factor is equal to ρ).

2.2 Discretization Schemes

The financial models mentioned above, describe the dynamics of stock price and volatility driven by continuous stochastic processes. Unfortunately, numerical methods rely on discrete models, only. Hence, the major stage of a simulation performed on any computational device is to discretize a continuous time process to a discrete time model.

Euler-Maruyama Formulas. This method is based on approximation of integrals using the left-point rule [12]. Having in mind that we simulate S_t over the period $[0, T]$, we can discretize this interval as $0 < t_1 < t_2 < ... < t_m = T$ where the difference between the subsequent t_k and t_{k+1} is equal to dt. Integrating the stochastic differential equation:

$$dS(t) = \alpha S(t)dt + \sigma(t)S(t)dW(t) \tag{7}$$

we obtain:

$$S_{t+dt} = S_t + \int_t^{t+dt} \alpha S(u)du + \int_t^{t+dt} \sigma(u)S(u)dW(u) \ , \tag{8}$$

where $\int_t^{t+dt} \alpha S(u)du \approx \alpha S(t) \int_t^{t+dt} du = \alpha S(t)dt$. Using the left-point rule since at time t the value $S(t)$ is known. The right-hand rule would require that $S(t+1)$ to be known at time t. In a similar fashion, the second integral might be approximated.

$$\int_t^{t+dt} \sigma(u)S(u)dW(u) \approx \sigma(t)S(t) \int_t^{t+dt} dW(u) = \sigma(t)S(t)(W_{t+dt} - W_t) = \sigma(t)S(t)\sqrt{dt}Z \tag{9}$$

where $W_{t+dt} - W_t$ and $\sqrt{dt}Z$ have an identical distribution (Z is a standard normal variable). Therefore, the discrete equation is as follows:

$$S_{t+dt} = S_t + \alpha S(t)dt + \sigma(t)S(t)\sqrt{dt}Z \tag{10}$$

Using these schemes, the recursive equations for the Heston Model might be obtained. The discrete formulae are as follows:

$$\begin{aligned} S_{t+1} &= S_t e^{(r-\frac{1}{2}\sigma_t)dt+\sqrt{\sigma_t dt}\cdot Z_1} \\ \sigma_{t+1} &= \sigma_t + \kappa \cdot (\theta - \sigma_t)dt + \xi \cdot \sqrt{\sigma_t dt} Z_2 \end{aligned}$$

2.3 Monte Carlo Simulation

Most problems in financial engineering entail the evaluation of a particular function involving multi-variable integrals with a high level of computational complexity. Unfortunately, in many cases these integrals cannot be evaluated in an efficient and analytic way. One of the methods dealing with these problems is the Monte Carlo simulation. This method is the most popular and efficient approach for determining the results of functions too complicated to being solved analytically. It comes from the fact, that the time taken to perform the Monte Carlo simulation increases approximately linearly with the number of random-samples, but other analytical routines tend to increase exponentially [7]. This estimation process is used to compute correct prices for financial options.

As only functions dependent on random variables are considered, its fundamentals are based on the weak law of large numbers. Estimation of the input function converges to the true likelihood as the number of generated random variables increases. The standard algorithm (the plain method) is to generate many random samples and evaluate the stochastic functions, whose results are further averaged. In this manner, we will converge to the correct results as the number of computed functions increases.

For financial models describing the evolution of a stock price or volatility, the Monte Carlo method involves many possible scenarios. In detail, each scenario denotes the sample pay-off calculated and discounted at the risk-free interest rate: $\Phi(S_{t,\theta}) = e^{-rT}(S_T - K)^+$. The expected value of the option price is equal to average of all the discounted payoffs, as follows:

$$v_{M,\theta} = \mathrm{E}(\Phi(S_\theta)) \approx \frac{\sum_{i=0}^{i=M} \Phi(S_{t,\theta})}{M} \tag{11}$$

Moreover, if we perform elementary mathematical operations for the expected value, we must take into account all the sample paths. For example, let us consider the differentiation:

$$\frac{dv_{M,\theta}}{d\theta} = \mathrm{E}(\frac{d\Phi(S_\theta)}{d\theta}) \approx \frac{\sum_{i=0}^{i=M} \frac{d\Phi(S_{t,\theta})}{d\theta}}{M} \; . \tag{12}$$

These values (known as the Greeks or Option Price Sensitivities) are very important measures of the potential financial risk related to a portfolio of derivatives.

2.4 Option Price Sensitivities

Introduction. Most of the financial institutions, selling or buying options from a client in the market, have to deal with problems of risk management. Aiming at a revenue growth, financial companies have to constantly balance its portfolio and neutralize risk of trading. In practice, transactions costs make frequent re-balancing very expensive. Except for trying to eliminate all risks, option traders focus on assessing risk and deciding whether it is acceptable. For hedging financial instruments, traders often evaluate rates of change of further predicted stock-price with different input parameters. These studies are to examine different scenarios analysis. In this method, the impact on the option position of alternative future scenarios is evaluated. As one could expect, rate of change of some function with respect to the input parameter converges to a derivative (when change of input parameter decreases). Hence, this approach is usually considered as a problem of evaluating derivatives. Financiers tend to use these values to quantify the different aspects of the risk inherent in their option portfolio. If the risk is acceptable, no adjustment is made to the portfolio, if it is unacceptable, quantitative analysts take an appropriate position for either the underlying asset or contract [7].

In practice, the second-order derivatives can be used, as well. Moreover, these values can be obtained in different manners which feature a various computational effort and a various accuracy. In the next paragraph, there is discussed a powerful approach - the Automatic Differentiation algorithms. One of them, the Adjoint method is the most accurate and much more computationally-efficient than existing solutions.

Automatic Differentiation. In particular, an underlying method for obtaining the first and the second-order derivatives is a combination of symbolic differentiation and automatic differentiation. This approach allows to evaluate the exact derivatives in the absence of round-off errors (further details are described in [9,14]). However, considering the symbolic solutions for Stochastic Differential Equations, we can state that evolution of the recursive formulas is usually long and dependent on a particular parameter - time. In this case, the AD methods should operate on the whole Kantorovich graph taking into account recursive dependencies. Nevertheless, this approach is memory-demanding. Therefore, we should rely only on a chain-rule as a dependency of subsequent variables, for example, S_i and S_{i+1} or σ_i and σ_{i+1}. By differentiating these functions we derive recursive equations for derivatives. Therefore, the sensitivities are dependent only on the previous computed sensitivities and some partial derivatives. The partial derivatives can be obtained by Forward or Reverse Mode, whereas, the sensitivities are previously equal to the initial statement and they further are correctly updated through evolution with respect to time. The following table presents the recursive formulas for the option price sensitivities for the Heston model. For clarity, only some of the sensitivities are presented and only the Heston model is considered.

Table 1. Stock price

Sensitivity	Recursive formula
δ	$\frac{dS_{t+1}}{dS_0} = \frac{dF_t(S_t)}{dS_t} \cdot \frac{dS_t}{dS_0}$
σ_0	$\frac{dS_{t+1}}{d\sigma_0} = \frac{dF_t(S_t)}{dS_t} \cdot \frac{dS_t}{d\sigma_0} + \frac{dF_t(\sigma_t)}{d\sigma_0}$
κ	$\frac{dS_{t+1}}{d\kappa} = \frac{dF_t(\sigma_t)}{d\sigma_t} \cdot \frac{d\sigma_t}{d\kappa} + \frac{dF_t(S_t)}{dS_t} \cdot \frac{dS_t}{d\kappa}$
θ	$\frac{dS_{t+1}}{d\theta} = \frac{dF_t(\sigma_t)}{d\sigma_t} \cdot \frac{d\sigma_t}{d\theta} + \frac{dF_t(S_t)}{dS_t} \cdot \frac{dS_t}{d\theta}$
ξ	$\frac{dS_{t+1}}{d\xi} = \frac{dF_t(\sigma_t)}{d\sigma_t} \cdot \frac{d\sigma_t}{d\xi} + \frac{dF_t(S_t)}{dS_t} \cdot \frac{dS_t}{d\xi}$

In a similar way, all the second-order derivatives can be proved for other financial models, as well. The partial derivatives are evaluated in a backward manner by using the Adjoint algorithm (Table 1).

3 CUDA Technology

In recent years, performing computations on GPU becomes more and more popular. Programming using GPU API, like CUDA [1] or OpenCL [3] is more difficult than traditional CPU programming (see, e.g., [2]), but gives us the access to powerful computational resources.

Implementations of interval methods in this environment (see, e.g., [8] and the references therein, [5,10]) show a significant performance gain.

Monte Carlo methods are also, specifically appropriate for GPU implementations as they perform the same procedure on several independent data (randomly chosen points).

Consequently, a proper implementation of option pricing algorithms on CUDA (or other GPU API) should be very efficient.

CUDA Random Library. Generating random realizations of varieties is an underlying process of the Monte Carlo method. For utilized here the CUDA technology, NVIDIA delivers a high performance GPU-accelerated library for random number generation. The CURAND library enables a wide variety of algorithms for random samples. The flexible model primarily allows to generate two fundamental types of random samples, pseudo-random and quasi-random numbers. The library offers several, most popular and truly random algorithms with a various distribution, among others: Mersenne Twister or XORWOW methods for pseudo-random samples or Sobol for quasi-random ones [4]. As CURAND consists of two fundamental modules, the interface to generate random numbers for the CPU side and kernel functions for GPU, it might be utilized for sequential implementation.

4 Architecture of the Library for Option Price Sensitivities

4.1 Introduction

It was crucial to effectively adapt Automatic Differentiation methods for the (first and second) derivatives of the recursive formulae (Euler-Maruyama scheme). An important feature was the ability to utilize not only various financial models or various discretization methods, but also different arithmetic and different data types. Another essential demand was a flexible, optimized implementation in terms of performance and lines of code. For this sake, the author designed own object-oriented software with a strong focus on a generic paradigm. For a simulation, it was decided to use a simple and relatively accurate method – the plain Monte Carlo algorithm. Different techniques of variance reduction will be considered as a further development.

The library was implemented in C++ and consists of the following modules:

- a template for arithmetic of different types (numbers, intervals, etc.), including overloaded operators,
- a template for financial models (Black Scholes, Heston, etc.),
- a generic code for the Monte Carlo methods – a template, initialized with financial model and data type during compilation,
- Automatic Differentiation functions.

Due to GPU limitations, an efficient, parallel implementation requires also some sophisticated solutions for problems of sequential nature. In order to reach a CUDA peak performance (to maximize throughput), an average evaluation and reduction of intermediate results are processed in a proper manner, dependent on the specific GPU architecture. Details can be found in [9].

4.2 Data Model

The Kantorovich Graph Representation. Kantorovich graphs [6] are used to represent the mathematical formulae, resulting from integrals' discretization.

The most important class – `Expression` – represents an independent, single term of a mathematical formula. Thanks to overloaded operators, the program dynamically generates the Kantorovich graph representation for every analytical model. This model is stored as a vector of nodes. This array is initialized in the right order, overloaded operators assign the Kantorovich graph with the priority of the subsequent operations. As the evaluation is performed starting from independent variables through intermediate to the last one, there is no need to sort the array in advance.

Each Node is an abstract structure representing a single operation consisting of the following fields: `left` and `right` indicators to preceding nodes (possibly empty) and `operationType` (exp, sin, variable, etc.). Unary operators utilize the structures for binary terms (the right indicator is null).

Representation of a Path. The Monte Carlo simulation requires storage of relatively large amount of generated paths, which might be difficult. Fortunately, combination of symbolic and automatic differentiation methods allows us to store and operate only a small number of parameters. For evaluation and further differentiation of a single path, we need as many temporary results as the number of nodes per the Kantorovich graph. Each node corresponds to a pair or a triple of the following variables: the value and one or two derivatives.

After computation of underlying asset and volatility for a single step (evolution with respect to time), these temporary values are updated. This approach allows not only to evaluate volatility or underlying asset, but differentiate these formulas according to the Forward Mode or the Reverse Mode algorithms. Additionally, a representation of a path supports a generic programming model. All the values might be single- or double-precision numbers as well as intervals.

4.3 Parallel Approach – GPU

Different path are processed in parallel, by different threads. The first stage of the parallel algorithm is the transfer of the Kantorovich graph representing both volatility and underlying asset. Besides that, additional structures are prepared and initialized in global memory, e.g., input parameters of an option or structures for storing the final derivatives. All Kantorovich graphs and computation results are stored in shared memory of a thread-block. Each thread is responsible for an evaluation and differentiation of a single path. Representations of mathematical models are distributed between thread-blocks. It seems an optimal approach in terms of performance, but current architectures do not allow to utilize all its horsepower. The number of threads within a thread-block depends on the financial model and shared-memory constraints per thread-block. Nevertheless, hundreds of paths are evaluated per thread-block and hundreds of thousands paths per grid to gain performance boost.

First, all threads should perform parallel initialization of the random generator. Afterwards, during the second kernel execution, the appropriate algorithm is processed. Then, each thread performs the following sequence of operations:

- generation of random samples with standard normal distribution (for i-th step of evolution through time),
- evaluation of the volatility and the underlying asset (for the i-th step of evolution through time),
- differentiation of the volatility and the underlying asset (for i-th step of evolution through time),
- updating the volatility and underlying assets ($\sigma_i \leftarrow \sigma_{i+1}$ and $S_i \leftarrow S_{i+1}$),
- updating the final sensitivities, relying on the results (as above) and well-known initial conditions.

After evaluation of maturities and the final derivatives for each path, all the results are averaged according to the optimized reduction algorithm. Using stream processing, we perform simultaneous data-transfer of some data and generation of random samples for another one. Details can be found in [9].

5 Computational Experiments

Computational experiments include studies of acceleration and uncertainty estimation for the Monte Carlo simulation and examine both sequential and parallel algorithm versions. Besides an evaluation of option prices, all the 1st-order derivatives are computed by using the Adjoint method. Analysis of performance, variance and rounding error have been investigated for Black-Scholes and Heston models. Tests were performed with a various range of paths and steps for maturity time. For generating random samples the Mersenne-Twister algorithm has been applied – implemented in the CURAND (GPU) and GNU Scientific Library (CPU).

The sequential version has been tested on a single CPU machine with Intel Xeon CPU X5650 and 48 GB RAM memory. The parallel one was run on NVIDIA Tesla C2070 with 48 multiprocessors and 448 cores, respectively.

6 Performance Results

Analysis of the acceleration for a parallel version (using shared memory) was compared to the CPU version. The vertical axis represents the ratio of run-time of parallel version to the sequential one. The horizontal axis describes a various number of sample paths for the Monte Carlo simulation. The time results for GPU take into account kernel initialization, data-transfer and kernel execution.

Black-Scholes Model. Much smaller memory requirements for the Black-Scholes model (only 13 nodes are necessary to store equations) allow to utilize a greater number of threads within a thread block (about 461). As one could expect, the speedup is proportional to the number of steps – evolution through time. For this reason, the results for the simulation with greater number of steps (300 steps) seem slightly better (simulation is 30 percents faster) (Tables 2, 3 and Fig. 1).

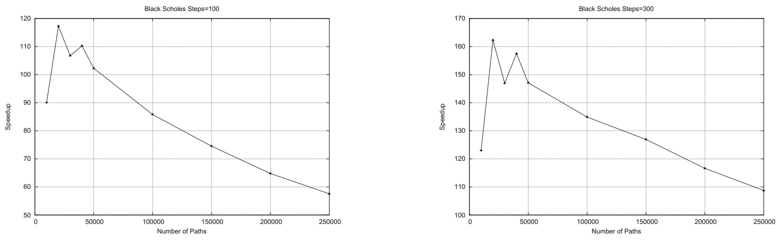
Heston Model. In contrast to the Black-Scholes model, the Heston equations are much more complex and memory-demanding (29 nodes for two equations representing stock price and volatility). Due to memory constraints of the Fermi architecture for shared memory, a slightly smaller number of threads within a thread-block is used (about 206 paths/per thread block). Considering the given plots, we observe a negligible increase of the acceleration over the first 200,000

Table 2. Black-Scholes, steps = 100

Nr_{PATHS}	$T_{CPU}[ms]$	$T_{GPU}[ms]$	Speedup
10000	615	6	90x
20000	1232	10	117x
50000	3083	30	102x
100000	6156	71	85x

Table 3. Black-Scholes, steps = 300

Nr_{PATHS}	$T_{CPU}[ms]$	$T_{GPU}[ms]$	Speedup
10000	1822	14	123x
20000	3653	22	162x
50000	9128	62	147x
100000	18264	135	134x



(a) Steps = 100 (b) Steps = 300

Fig. 1. Parallel approach for CUDA – Black-Scholes model

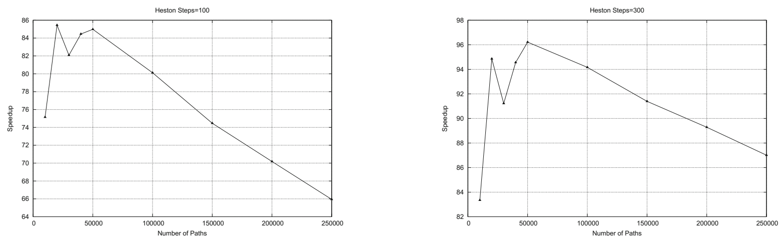
Table 4. Heston, steps = 100

Nr_{PATHS}	$T_{CPU}[ms]$	$T_{GPU}[ms]$	Speedup
10000	1537	20	75x
20000	3070	35	85x
50000	7684	90	84x
100000	15374	191	80x

Table 5. Heston, steps = 300

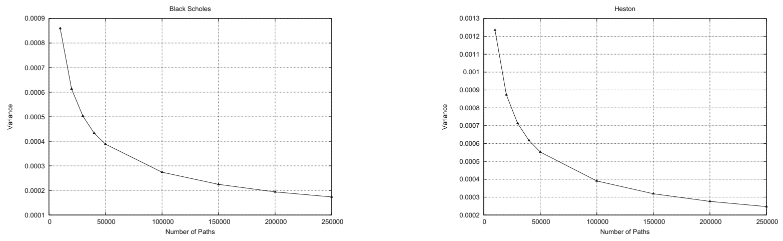
Nr_{PATHS}	$T_{CPU}[ms]$	$T_{GPU}[ms]$	Speedup
10000	4600	55	83x
20000	9199	96	94x
50000	22981	238	96x
100000	45995	488	94x

paths. For 25000 paths, the line chart reaches the performance peak (96x). With the increasing number of paths, we see a steady and smooth decrease of the acceleration. This decline is motivated by data-transfer of large number of paths and the derivatives via PCI-Express bus (Tables 4, 5 and Fig. 2).



(a) Steps = 100 (b) Steps = 300

Fig. 2. Parallel approach for CUDA – Heston model



(a) Black-Scholes Model (b) Heston Model

Fig. 3. Variance results – option price

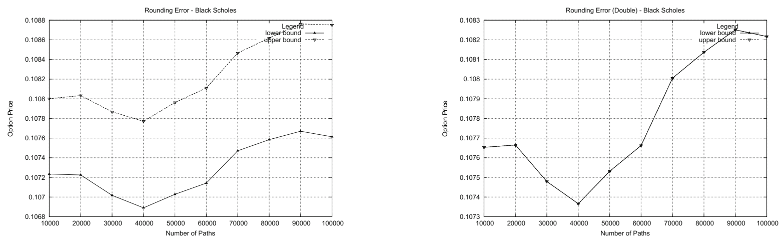
6.1 Estimation Uncertainty Results

The uncertainty tests were conducted on a various number of paths. The variance results are evaluated for stock price and the delta sensitivity (the first-order derivative of $\frac{dS_T}{dS_0}$).

Black-Scholes Model. With our expectations, change (decrease) of variance is proportional to a square of the number of sample paths. Initially, the variance stood at the level of 10^{-4}, however, the sudden collapse might be observed over next 50000 paths. Starting from 100,000 paths, the increasing number of sample paths has an insignificant impact on change of the Monte Carlo estimator. Finally, the variance is approximately equal to $1.7 * 10^{-5}$ for 250,000 paths.

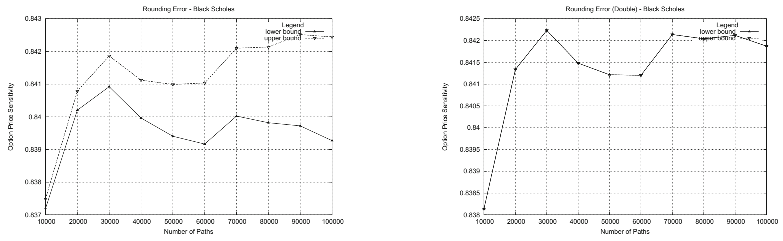
Heston Model. For the Heston model simulation, the variance measure is slightly bigger than variability for the Black-Scholes. As expected, the line chart has a declining trend and the variance is dependent on the number of sample paths (the factor is proportional to square of the number of paths) (Fig. 3).

Rounding Error Analysis. The Monte Carlo simulation requires several arithmetic operations throughout path evolution, which may increase the rounding error. To manage this issue, we used the interval analysis (see, e.g., [11]). For Monte-Carlo simulations, interval operators are applied to unary and binary



(a) single-precision numbers (b) double-precision numbers

Fig. 4. Black-Scholes model – option price



(a) single-precision numbers (b) double-precision numbers

Fig. 5. Black-Scholes model – option price sensitivity

calculations for all the paths that are required to evaluate the option price and its sensitivities, bounding the errors.

We investigated both single and double-precision numbers. For single-precision, rounding errors were significant – diameters of resulting intervals increase with the number of paths. Using double-precision numbers, the rounding error occurred to be negligible, at least for the considered case-studies (Figs. 4, 5).

7 Conclusion

Automatic Differentiation is a powerful tool that can be useful for evaluating option prices and the Greeks and modern GPUs allow its efficient implementations. The paper investigated two common models – Black-Scholes and Heston. Expectations of stochastic processes have been computed using Monte Carlo methods. Using interval arithmetic allowed to bound numerical errors and use imprecise parameters. The approach might be applicable also to calibrate theoretical financial models to market quotes. The interval approach might be combined with other global optimization methods into the calibration process, which is going to be subject of future research.

References

1. CUDA homepage. http://www.nvidia.com/object/cuda_home.html
2. Nvidia, CUDA SDK Documentation. http://docs.nvidia.com/cuda/index.html
3. OpenCL homepage. http://www.khronos.org/opencl
4. Nvidia, CUDA CURAND Library. https://developer.nvidia.com/curand
5. Beck, P.-D., Nehmeier, M.: Parallel interval newton method on CUDA. In: Manninen, P., Öster, P. (eds.) PARA 2012. LNCS, vol. 7782, pp. 454–464. Springer, Heidelberg (2013)
6. Bücker, M.: Automatic Differentiation: Applications, Theory and Implementation. Springer, Berlin (1981)
7. Hull, J.C.: Options, Futures and other Derivatives, 8th edn. Prentice Hall, Upper Saddle River (2011)
8. Kozikowski, G.: Implementation of automatic differentiation library using the OpenCL technology. BEng thesis, Faculty of Electronics and Information Technology, WUT (2011)
9. Kozikowski, G.: Evaluation of option price sensitives based on the Automatic Differentiation methods using CUDA. Master's Thesis, Faculty of Electronics and Information Technology, WUT (2013)
10. Kozikowski, G., Kubica, B.J.: Interval arithmetic and automatic differentiation on GPU using OpenCL. In: Manninen, P., Öster, P. (eds.) PARA 2012. LNCS, vol. 7782, pp. 489–503. Springer, Heidelberg (2013)
11. Kubica, B.J.: A class of problems that can be solved using interval algorithms. Computing **94**(2–4), 271–280 (2012). (SCAN 2010 proceedings)
12. Rouah, F. D.: Euler and Milstein Discretization. http://www.frouah.com/finance%20notes/Euler%20and%20Milstein%20Discretization.pdf
13. Tadjouddine, E.M., Cao, Y.: An option pricing model calibration using algorithmic differentiation. In: Gelenbe, E., Lent, R., Sakellari, G. (eds.) Computer and Information Sciences II, pp. 577–581. Springer, London (2012)
14. Werbos, P.: Backpropagation through time: what it does and how to do it. Proc. IEEE **78**, 1550–1560 (1990)

Subsquares Approach – A Simple Scheme for Solving Overdetermined Interval Linear Systems

Jaroslav Horáček(✉) and Milan Hladík

Faculty of Mathematics and Physics, Department of Applied Mathematics,
Charles University, Prague, Czech Republic
{horacek,hladik}@kam.mff.cuni.cz

Abstract. In this work we present a new simple but efficient scheme – Subsquares approach – for development of algorithms for enclosing the solution set of overdetermined interval linear systems. We are going to show two algorithms based on this scheme and discuss their features. We start with a simple algorithm as a motivation, then we continue with an improved algorithm. Both algorithms can be easily parallelized. The features of both algorithms will be discussed and numerically tested.

Keywords: Interval linear systems · Interval enclosure · Overdetermined systems · Parallel computing

1 Introduction

In this paper we address the problem of solving overdetermined interval linear systems (OILS). They can occur in many applications, e.g., computing eigenvectors of interval matrices [2] or when solving various continuous CSP problems. There exist a lot of efficient methods for solving square interval linear systems. Solving overdetermined systems is a little bit more tricky, that is because we can not use some favourable properties of matrices like diagonal dominance, positive definiteness, etc. Nevertheless, there are some methods – Rohn method [8], linear programming [3], Gaussian elimination [1] or the method designed by Popova [7].

Some of the methods return narrow enclosures of a solution set. But they return a solution even if the system is unsolvable. Other methods often rely on some kind of preconditioning which leads to enclosure overestimation and for some systems (e.g., those with wide intervals) can not be done. It is very difficult to develop one method suitable for all types of systems. We would like to present a scheme – Subsquares approach – which enables us to develop methods for solving overdetermined interval linear systems. We will derive a simple method according to this scheme. Then, we will derive an improved method. Both are suitable for parallel computing. Before introducing the scheme and derived methods, it would be desirable to start with some basic interval notation and definitions first.

2 Basics of Interval Arithmetics

In this text we work with closed real intervals $\boldsymbol{x} = [\underline{x}, \overline{x}]$, where $\underline{x} \leq \overline{x}$. The numbers $\underline{x}, \overline{x}$ are called the lower bound and upper bound respectively.

We will use intervals as coefficients of matrices and vectors during our computations. The interval representation may be useful in many ways – it may represent uncertainty (e.g., lack of knowledge, damage of data), verification (e.g., errors in measurement), computational errors (e.g., rounding errors in floating point arithmetic), etc. Intervals and interval vectors will be denoted in boldface, i.e., $\boldsymbol{x}, \boldsymbol{b}$. Interval matrices will be denoted by bold capitals, i.e., $\boldsymbol{A}$, $\boldsymbol{C}$.

Another notion we will use is the *midpoint* of an interval $\boldsymbol{x}$, it is defined as $x_c = (\underline{x}+\overline{x})/2$. By A_c we will denote the midpoint matrix of $\boldsymbol{A}$. When comparing two intervals we need the notion *width* of an interval $\boldsymbol{x}$ defined as $\mathrm{w}(\boldsymbol{x}) = \overline{x} - \underline{x}$. If $\boldsymbol{u}$ is an n-dimensional interval vector we will define "width" and "volume" of $\boldsymbol{u}$ as

$$\mathrm{W}(\boldsymbol{u}) = \sum_{i=1}^{n} \mathrm{w}(\boldsymbol{u}_i), \quad \mathrm{V}(\boldsymbol{u}) = \prod_{i=1}^{n} \mathrm{w}(\boldsymbol{u}_i),$$

respectively.

Vector and matrix operations will be defined using the standard interval arithmetic, for definition of interval operations and further properties of the interval arithmetic do not hesitate to see e.g. [5].

We continue with definitions connected with interval linear systems. Let us have an interval linear system $\boldsymbol{A}x = \boldsymbol{b}$, where $\boldsymbol{A}$ is an $m \times n$ matrix. When $m = n$ we will call it a *square* system. When $m > n$ we will call it an *overdetermined* system. In the further text, when talking about an overdetermined system, we will always use the notation $\boldsymbol{A}x = \boldsymbol{b}$, where $\boldsymbol{A}$ is an $m \times n$ matrix.

It is necessary to state what do we mean by the solution set of an interval linear system. It is the set

$$\Sigma = \{\, x \mid Ax = b \text{ for some } A \in \boldsymbol{A}, b \in \boldsymbol{b} \,\}.$$

We shall point out, that it is different from the interval least squares solution set which is defined as

$$\Sigma_{lsq} = \{\, x \mid A^T Ax = A^T b \text{ for some } A \in \boldsymbol{A}, b \in \boldsymbol{b} \,\}.$$

If a system has no solution, we call it *unsolvable*. The *interval hull* is an n-dimensional box (aligned with axes) enclosing the solution set as tightly as possible. When we start using intervals in our computations (we have mentioned its advantage already), many problems become NP-hard [4]. So is the problem of finding the hull of the solution set [10]. It can be computed quite painfully using, e.g., linear programming [3]. That is why we are looking for a little wider n-dimensional box containing the hull. The tighter the better. We call it interval *enclosure*. In an introduction section we named some methods for computing interval enclosures of Σ set. It can be proved that $\Sigma \subseteq \Sigma_{lsq}$. Therefore, we can also use methods for computing the interval least squares as methods for computing interval enclosures [3]. The routine `verifylss` from Matlab interval toolbox Intlab uses this kind of approach. Because of using the interval least squares this routine returns solution even if the whole system is unsolvable.

In this work we will provide numerical testing at various places in the text, therefore we rather mention its parameters here. The testing will be done on AMD Phenom II X6 1090T 3200 MHz, with 15.5 GB memory. We used Matlab R2012b with toolbox for interval computation INTLAB v6 [11] and Versoft v10 [9] for verified interval linear programming.

All examples will be tested on random overdetermined systems. A random system is generated in the following way. First, we generate a random solvable point overdetermined system. Coefficients are taken uniformly from interval $[-20, 20]$. Then, we inflate the coefficients of this system to intervals of certain width. The original point system is not necessarily a midpoint system of the new interval system. Each of its coefficients is randomly shifted towards one of the bounds of an interval in which it lies.

3 Subsquares Approach

By a square subsystem (we will also call it a *subsquare*) of an overdetermined system we mean a system composed of some equations taken (without repetition) from the original overdetermined system such that together they from a square system. Some of the possibilities are shown in the Fig. 1. The rows represent equations. For the sake of simplicity we will denote the square subsystem of $\boldsymbol{A}x = \boldsymbol{b}$ created by equations $i_1, i_2, \ldots, i_n$ as $\boldsymbol{A}_{\{i_1,i_2,\ldots,i_n\}}x = \boldsymbol{b}_{\{i_1,i_2,\ldots,i_n\}}$. When we use some order (e.g., dictionary order) of subsquares (here it does not depend which one) the j-th system will be denoted $\boldsymbol{A}_j x = \boldsymbol{b}_j$.

Let us suppose we can solve a square interval system efficiently and quickly. We can take for example one of the following method – Jacobi method [5], Gauss-Seidel method [5,6], Krawczyk method [5,6], etc. These methods usually can not be applied to overdetermined systems. Nevertheless, we can use the fact that we can solve the square systems efficiently together with the fact that the solution set of an overdetermined interval system must lie inside the solution set of its arbitrary subsquare. This follows from the fact that by removing some equations from an overdetermined system we can only make the solution set of the smaller system equal or larger (because we removed some restrictions).

When we chose some subsquares of an overdetermined system we can simply provide an intersection of their solution enclosures or provide some further work. We get a simple algorithm for solving overdetermined interval linear systems. As a motivation for this approach let us take the randomly generated interval system $\boldsymbol{A}x = \boldsymbol{b}$ (with rounded bounds), where

$$\boldsymbol{A} = \begin{bmatrix} [\ -0.8, & 0.2] & [-20.1, & -19.5] \\ [-15.6, & -15.2] & [\ 14.8, & 16.7] \\ [\ 18.8, & 20.1] & [\ 8.1, & 9.5] \end{bmatrix}, \qquad (1)$$

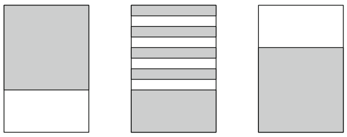


Fig. 1. Various square subsystems

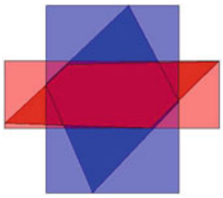


Fig. 2. Solution sets and hulls of subsquares (Color figure online)

$$\boldsymbol{b} = \begin{bmatrix} [\ \ 292.1, & \ \ 292.7\ ] \\ [\ -361.9, & -361.1\ ] \\ [\ \ \ 28.4, & \ \ \ 30.3\ ] \end{bmatrix}. \tag{2}$$

In the Fig. 2 we can see the solution set and the hull of $\boldsymbol{A}_{\{1,2\}}x = \boldsymbol{b}_{\{1,2\}}$ (red color) and the same for $\boldsymbol{A}_{\{2,3\}}x = \boldsymbol{b}_{\{2,3\}}$ (blue color). It can be seen that if we provide intersection of the two hulls (or enclosures), the resulting enclosure of the $\boldsymbol{A}x = \boldsymbol{b}$ solution set might get remarkably tighter.

Every method choosing subsquares from an overdetermined system and then providing some further work over these subsquares we call *subsquares method.* The principle of computing enclosure of solution set this way we call *subsquares approach.*

3.1 Simple Algorithm

If we compute enclosures of square subsystems separately and then intersect resulting enclosures, we get the simple Algorithm 1 for solving OILS.

Algorithm 1. Subsquares method – simple algorithm

```
Require: $\boldsymbol{A}, \boldsymbol{b}$
Ensure: enclosure $\boldsymbol{x}$ of the solution set of $\boldsymbol{A}x = \boldsymbol{b}$
  $\boldsymbol{x} = [-\infty, \infty]^n$
  while not (terminal condition) do
    choose randomly a subsquare of $\boldsymbol{A}x = \boldsymbol{b}$
    compute its enclosure $\boldsymbol{x}_{subsq}$
    $\boldsymbol{x} := \boldsymbol{x} \cap \boldsymbol{x}_{subsq}$
  end while
```

Table 1. Simple subsq. method solving all subsquares – enclosures comparison

System	$\mathrm{av}\left(\frac{\mathrm{W}(\boldsymbol{x}_{subsq})}{\mathrm{W}(\boldsymbol{x}_{hull})}\right)$	$\mathrm{av}\left(\frac{\mathrm{V}(\boldsymbol{x}_{subsq})}{\mathrm{V}(\boldsymbol{x}_{hull})}\right)$	$\mathrm{av}\left(\frac{\mathrm{W}(\boldsymbol{x}_{ver})}{\mathrm{W}(\boldsymbol{x}_{hull})}\right)$	$\mathrm{av}\left(\frac{\mathrm{V}(\boldsymbol{x}_{ver})}{\mathrm{V}(\boldsymbol{x}_{hull})}\right)$
5×3	1.0014	1.0043	1.1759	1.6502
9×5	1.0028	1.0140	1.1906	2.3831
13×7	1.0044	1.0316	1.2034	3.6733
15×9	1.0061	1.0565	1.1720	4.2902
25×21	1.0227	1.6060	1.0833	5.4266
30×29	1.0524	5.8330	1.0987	51.0466

Table 2. Simple subsq. method – unsolvability detection

System	$rad = 0.01$	$rad = 0.001$	$rad = 0.0001$
15×10	2.1	2.0	2.0
25×21	2.2	2.0	2.0
35×23	2.2	2.0	2.0
50×35	2.4	2.0	2.0
73×55	2.9	2.1	2.0
100×87	7.1	2.1	2.0

This approach is a little bit naive, but it has its advantage. First, if we compute enclosures of all possible square subsystems, we get really close to the interval hull. The Table 1 shows the average ratios of widths and volumes of enclosures $\boldsymbol{x}_{subsq}$, $\boldsymbol{x}_{ver}$ returned by simple subsquares method and `verifylss` compared to the interval hull $\boldsymbol{x}_{hull}$ computed by linear programming. If we have an $m \times n$ system, the number of all square subsystems is equal to $\binom{m}{n}$. However, we can see that for n small or for n close to m the number $\binom{m}{n}$ might not be so large. That is why solving all subsquares pays off when systems are tall or nearly-squared.

The second advantage is that Algorithm 1 can, in contrast to other methods, often decide whether a system is unsolvable – if, in some iteration, the resulting enclosure is empty after intersection, then the overdetermined system is unsolvable. The Table 2 shows average number of random subsquares chosen until the unsolvability was discovered (empty intersection occurred). Each column represents systems of different coefficient radii. We can see that for systems with relatively small intervals unsolvability was revealed almost immediately.

For most rectangular systems it is however not convenient to compute enclosures of all or many square subsystems. The choice of subsquares and the solving algorithm can be modified to be more economical and efficient.

3.2 Improved Algorithm

We wish to have a method that returns sharp enclosures, can reveal unsolvability and is parallelizable. All can be done by the simple algorithm. However, there is a problem – extremely long computation time for a general system.

We would like to design an improved method, still using subsquares approach, that can reduce the computation time and can work for more general cases. First idea is choosing better strategy to "get together" various enclosures of subsquares, not only intersection. We need an algorithm that allows to propagate newly computed information faster. Second idea is selecting only some subsquares, not all of them.

About the first idea. When we talk about immediate propagation of partially computed solution, our mind can easily come to Gauss-Seidel iterative method (GS). This method works for square interval systems. Let us have a square system $\boldsymbol{C}x = \boldsymbol{d}$. The method starts with an initial enclosure $\boldsymbol{x}^{(0)}$. In k-th iteration each entry of the current enclosure vector $\boldsymbol{x}^{(k-1)}$ might be narrowed using the formula

$$\begin{aligned}\boldsymbol{x}_i^{(k)} = \frac{1}{\boldsymbol{C}_{ii}}\Big[\ \boldsymbol{d}_i - (\boldsymbol{C}_{i1}\boldsymbol{x}_1^{(k)} + \ldots + \boldsymbol{C}_{i(i-1)}\boldsymbol{x}_{i-1}^{(k)} + \\ +\ \boldsymbol{C}_{i(i+1)}\boldsymbol{x}_{i+1}^{(k-1)} + \ldots + \boldsymbol{C}_{in}\boldsymbol{x}_n^{(k-1)})\ \Big] \cap\ \boldsymbol{x}_i^{(k-1)}.\end{aligned}$$

Simply said, in each iteration this algorithm expresses $\boldsymbol{x}_i$ from i-th equation of $\boldsymbol{A}x = \boldsymbol{b}$ and intersects with the old value. It uses the newly computed values immediately.

In our algorithm we will use GS iteration in a similar way for more square subsystems simultaneously. Again, we start with some initial enclosure $\boldsymbol{x}^{(0)}$. In k-th iteration we provide k-th GS iteration step for all systems. The advantage of this technique is that we express each variable according to formulas originating from more systems. We expect the narrowing rate will be much better this way. Similarly as in simple GS, if in some point of computation empty intersection occurs, whole overdetermined system has no solution.

Iterative methods usually require a preconditioning. We will use the preconditioning with A_c^{-1}. There are still two yet not answered problems – initial enclosure and terminal condition. To find $\boldsymbol{x}^{(0)}$, we can take the resulting enclosure of some other method. Or we can compute an enclosure of one square subsystem. The algorithm will terminate after k-th iteration if e.g.

$$\forall\, i \quad \left|\underline{\boldsymbol{x}}_i^{(k)} - \underline{\boldsymbol{x}}_i^{(k-1)}\right| < \epsilon \quad \text{and} \quad \left|\overline{\boldsymbol{x}}_i^{(k)} - \overline{\boldsymbol{x}}_i^{(k-1)}\right| < \epsilon,$$

for some small positive ϵ and $i = 1, \ldots, n$.

About second idea. Now we would like to choose some subsystems. Here are some desirable properties of the set of selected subsquares:

1. We do not want to have too many subsquares
2. We want to cover the whole overdetermined system by some subsquare
3. The overlap of subsquares is not too low, not too high
4. We take subsquares that narrow the resulting enclosure as much as possible

We can select subsquares randomly, but then we do not have the control over this selection. This works fine, however, it is not clear how many subsquares should we choose according to the size of the overdetermined system. Moreover, experiments have shown that it is advantageous when subsquares overlap. That is why we propose a different strategy.

First and second property can be solved by covering the system step by step using some overlap parameter. About third property, experiments show that taking overlap $\approx n/3$ is a reasonable choice. Property four is a difficult task to provide. We think deciding which systems to choose (in a favourable time) is still an area to be explored. Yet randomness will serve us well. Among many possibilities we tested, the following selection of subsystems worked well. During our algorithm we divide numbers of equations of an overdetermined system into two sets – $Covered$, which contains equations that are already contained in some subsystems, and $Waiting$, which contains equations that are not covered yet. We also use a parameter *overlap* to define the overlap of two subsequent subsquares.

The first subsystem is chosen randomly, other subsystems will be composed of *overlap* equations with indices from $Covered$ and $n - overlap$ equations with indices from $Waiting$. The last system is composed of all remaining uncovered equations and then some already covered equations are added to form a square system. This has described the Algorithm 2. The algorithm is not necessarily optimal, it should serve as an illustration. The procedure *randsel(n, S)* selects n random non-repeating numbers from the set S. The total number of subsquares chosen by this algorithm is $1 + \left\lceil \frac{m-n}{n-overlap} \right\rceil$.

The whole improved algorithm is summarized as Algorithm 3. The function *GS-iteration(* $\boldsymbol{C}x = \boldsymbol{d}$, $\boldsymbol{y}$*)* applies one iteration of Gauss-Seidel method on the (already preconditioned) subsquare $\boldsymbol{C}x = \boldsymbol{d}$ using $\boldsymbol{y}$ as initial enclosure. Method *has-converged()* returns true if terminal condition (e.g., the one mentioned earlier) is satisfied.

As we showed in [3], `verifylss` (using the interval least squares approach) from INTLAB is one of the best and quite general method for overdetermined systems that are solvable. That is why we wanted to compare our method with this method. During our testing we realized `verifylss` works fine with small intervals, however it is not too efficient when the intervals become relatively large. We used enclosures returned by `verifylss` as inputs for the improved method and tested if our algorithm was able to narrow them. The Table 3 shows the results. Column *rad* shows radii of intervals of testing systems, we chose the same radii for all coefficients of a system. We tested on 100 random systems. For each system we chose 100 random subsquares sets and applied the improved method on them. The fourth column shows average ratios of enclosure widths of $\boldsymbol{x}_{subsq}$ and $\boldsymbol{x}_{ver}$. If the ratio is 1, then we were not able to sharpen $\boldsymbol{x}_{ver}$. The lower the number, the better narrowing of $\boldsymbol{x}_{ver}$ by the improved method. Each random selection of subsquares set usually produces a different ratio of enclosure widths. The fifths collumn shows standard deviation. For each system we chose one of the 100 subsquares sets that produces the best ratio. The sixth column shows the average value of the best ratios found for each of 100 random systems. Columns t_{ver} and t_{subsq} show computation times (in seconds) of `verifylss` and the improved method respectively.

Algorithm 2. Choosing square subsystems

```
Require: $\boldsymbol{A}, \boldsymbol{b}$, overlap
Ensure: set of subsquares of $\boldsymbol{A}x = \boldsymbol{b}$
  Systems ← $\emptyset$ {set of square subsystems}
  Covered ← $\emptyset$ {numbers of covered equations by some subsystem}
  Waiting ← $\{1, 2, \ldots, m\}$ {numbers of equations to be covered}
  Indices ← $\emptyset$ {numbers of equations of one subsquare}

  while Waiting $\neq \emptyset$ do
    if Covered $= \emptyset$ then
      Indices ← randsel($n$, Waiting)

    else if |Waiting| $\leq$ ($n$ − overlap) then
      Indices ← Waiting ∪ randsel($n$ − |Waiting|, Waiting)

    else
      Indices ← randsel(overlap, Covered) ∪ randsel(n−overlap, Waiting)
    end if

    Systems ← Systems ∪ $\{\boldsymbol{A}_{\text{Indices}}x = \boldsymbol{b}_{\text{Indices}}\}$
    Covered ← Covered ∪ Indices
    Waiting ← Waiting \ Indices
  end while
  return Systems
```

The larger interval radii are, the more intensively our method sharpens `verifylss` enclosures. When the interval radii are large it often happens that `verifylss` returns infinite enclosure but when we use a different initial enclosure and then apply our algorithm we get finite enclosure. The computation times of the improved method are longer, but still not very demanding. We are dealing with NP-hard problem and every improvement of enclosure (towards the hull) might be more and more painfull.

3.3 Parallel Algorithm

The naive algorithm can be easily parallelized. All the square subsystems can be solved in parallel and then all enclosures are intersected. We have only one barrier at the end of computation.

If we take a look at computation times in the Table 3, we realize `verifylss` is much faster. However, the time pay for gaining much precise enclosure using the improved method is not too high. Moreover, even the improved algorithm can be parallelized. Propagation of newly computed enclosure can be guaranteed by sharing the currently computed enclosure vector $\boldsymbol{x}$ among processors as a global variable. If we use Jacobi formula instead of Gauss-Seidel formula for one iteration, the computation becomes of kind SIMD - single instruction multiple data. Therefore it could be used even on GPUs – one pipeline narrows one

Algorithm 3. Subsquares method – improved version

```
Require: $\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{x}^{(0)}$
Ensure: enclosure $\boldsymbol{x}$ of the solution set $\boldsymbol{A}x = \boldsymbol{b}$
  select $k$ subsquares $\{\boldsymbol{A}_1 x = \boldsymbol{b}_1, \ldots, \boldsymbol{A}_k x = \boldsymbol{b}_k\}$ by Algorithm 2
  $\boldsymbol{x} \leftarrow \boldsymbol{x}^{(0)}$
  converged $\leftarrow$ false
  while not converged do
    for $i = 1$ to $k$ do
      $\boldsymbol{x} \leftarrow$ GS-iteration($\boldsymbol{A}_i x = \boldsymbol{b}_i, \boldsymbol{x}$)
    end for
    converged $\leftarrow$ has-converged()
  end while
  return $\boldsymbol{x}$
```

Table 3. Subsquare method shaving the verifylss enclosure

System	Overlap	rad	av. rat	best av. rat	std	t_{ver}	t_{subsq}
15×10	3	0.01	0.9961	0.9677	0.0068	0.0043	0.0345
15×10	3	0.1	0.9778	0.8841	0.0316	0.0046	0.0472
15×10	3	0.25	0.8573	0.5851	0.1654	0.0071	0.0871
15×10	3	0.35	0.7495	0.3931	0.2285	0.0093	0.1287
25×13	5	0.01	0.9990	0.9862	0.0027	0.0045	0.0555
25×13	5	0.1	0.9924	0.9494	0.0126	0.0050	0.0770
25×13	5	0.25	0.9229	0.7111	0.0838	0.0089	0.1559
25×13	5	0.35	0.7563	0.3175	0.2366	0.0098	0.3476
37×20	7	0.01	0.9999	0.9964	0.0005	0.0065	0.0730
37×20	7	0.1	0.9964	0.9706	0.0063	0.0078	0.1109
37×20	7	0.25	0.8784	0.3197	0.1640	0.0147	0.4799
50×35	11	0.1	0.9708	0.7776	0.0478	0.0193	0.2939

variable from the interval enclosure vector, a bunch of pipelines computes over one subsquare. Nevertheless, shared vector $\boldsymbol{x}$ might create a bottleneck. We believe this could by prevented by the following behaviour of each pipeline. When reading, each pipeline does not lock corresponding shared variable. After each iteration it overwrites a shared variable only if it has better enclosure than the one currently stored there. This is inspired with an observation, that in one iteration not many computations over different subsquares improve the same variable. However, we assume that there exist much more efficient ways how to make parallel subsquares methods more efficient and memory collision avoiding.

4 Conclusion

In this paper we introduced a simple but efficient scheme – subsquares approach – for enclosing the solution set of overdetermined interval linear systems. The first method derived from this scheme was a little bit naive, but for tall or nearly-square systems it was able to find almost the interval hull. The second method was a little bit more sophisticated but still quite simple. It worked well on interval systems which coefficients were composed of wide intervals. This method was able to significantly sharpen enclosures produced by `verifylss` Intlab routine. Both methods were able to detect unsolvability of OILS. Moreover, they could be easily parallelized. In the second method we choose the square subsystems randomly, that is why sharpening produced by this method has variable results. There is an open question whether for each OILS there exists a deterministically chosen set of subsquares which gives the best possible enclosure, so we can avoid the randomization.

Acknowledgement. Our research was supported by the grant GAČR P402/13/10660S. Jaroslav Horáček was partially supported by the Grant Agency of the Charles University (GAUK) grant no. 712912 and by GAUK no. SVV-2013–267313. Jaroslav Horáček would like to thank to Jezci mix-research group for general support.

References

1. Hansen, E.R., Walster, G.W.: Solving overdetermined systems of interval linear equations. Reliable Comput. **12**(3), 239–243 (2006)
2. Hladík, M., Daney, D., Tsigaridas, E.P.: An algorithm for addressing the real interval eigenvalue problem. J. Comput. Appl. Math. **235**(8), 2715–2730 (2011)
3. Horáček, J., Hladík, M.: Computing enclosures of overdetermined interval linear systems. Submitted to Reliable Computing, text available at http://arxiv.org/abs/1304.4738 (2013)
4. Kreinovich, V., Lakeyev, A., Rohn, J., Kahl, P.: Computational Complexity and Feasibility of Data Processing and Interval Computations. Kluwer, Dordrecht (1998)
5. Moore, R.E., Kearfott, R.B., Cloud, M.: Introduction to Interval Analysis. Society for Industrial Mathematics, Philadelphia (2009)
6. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge University Press, Cambridge (1990)
7. Popova, E.D.: Improved solution enclosures for over- and underdetermined interval linear systems. In: Lirkov, I., Margenov, S., Waśniewski, J. (eds.) LSSC 2005. LNCS, vol. 3743, pp. 305–312. Springer, Heidelberg (2006)
8. Rohn, J.: Enclosing solutions of overdetermined systems of linear interval equations. Reliable Comput. **2**(2), 167–171 (1996)
9. Rohn, J.: VERSOFT: Verification software in MATLAB / INTLAB, version 10. http://uivtx.cs.cas.cz/~rohn/matlab/ (2009)
10. Rohn, J., Kreinovich, V.: Computing exact componentwise bounds on solutions of lineary systems with interval data is np-hard. SIAM J. Matrix Anal. Appl. **16**(2), 415–420 (1995)
11. Rump, S.M.: INTLAB - INTerval LABoratory. In: Csendes, T. (ed.), Developments in Reliable Computing, pp. 77–104. Kluwer Academic Publishers, Dordrecht (1999) http://www.ti3.tu-harburg.de/rump/

Using Quadratic Approximations in an Interval Method for Solving Underdetermined and Well-Determined Nonlinear Systems

Bartłomiej Jacek Kubica(✉)

Institute of Control and Computation Engineering, Warsaw University of Technology, Warsaw, Poland
bkubica@elka.pw.edu.pl

Abstract. This paper considers quadratic approximation as a narrowing tool in an interval branch-and-prune method. We seek the roots of such an approximate equation – a quadratic equation with interval parameters. Heuristics to decide, when to use the developed operator, are proposed. Numerical results for some benchmark problems are presented and analyzed.

Keywords: Nonlinear equations systems · Interval computations · Quadratic approximation · Interval quadratic equation · Heuristic

1 Introduction

In the paper [11] the author considered an interval solver for nonlinear systems – targeted mostly at underdetermined equations systems – and its shared-memory parallelization. In subsequent papers several improvements have been considered, including various parallelization tools [12] and using sophisticated tools and heuristics to increase the efficiency of the solver [13]. As indicated in [13] and [14], the choice of proper tools and the proper heuristic, for their selection and parameterization, appeared to have a dramatic influence on the efficiency of the algorithm.

2 Generic Algorithm

The solver uses interval methods.They are based on interval arithmetic operations and basic functions operating on intervals instead of real numbers (so that result of an operation on numbers belong to the result of operation on intervals, containing the arguments). We shall not define interval operations here; the interested reader is referred to several papers and textbooks, e.g., [8,9,19].

The solver is based on the branch-and-prune (B&P) schema that can be expressed by the following pseudocode:

```
IBP ($\mathbf{x}^{(0)}$; f)
+//+$\mathbf{x}^{(0)}$ is the initial box, f(·) is the interval extension of the function $f\colon \mathbb{R}^n \to \mathbb{R}^m$
// $L_{ver}$ is the list of boxes verified to contain a segment of the solution manifold
// $L_{pos}$ is the list of boxes that possibly contain a segment of the solution manifold
$L = L_{ver} = L_{pos} = \emptyset$ ;
$\mathbf{x} = \mathbf{x}^{(0)}$ ;
loop
    process the box $\mathbf{x}$, using the rejection/reduction tests ;
    if ($\mathbf{x}$ does not contain solutions) then discard $\mathbf{x}$ ;
    else if ($\mathbf{x}$ is verified to contain a segment of the solution manifold) then
        push ($L_{ver}$, $\mathbf{x}$) ;
    else if (the tests resulted in two subboxes of $\mathbf{x}$: $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$) then
        $\mathbf{x} = \mathbf{x}^{(1)}$ ;
        push ($L$, $\mathbf{x}^{(2)}$) ;
        cycle loop ;
    else if ($\mathbf{x}$ is small enough) then push ($L_{pos}$, $\mathbf{x}$) ;
    if ($\mathbf{x}$ was discarded or stored) then
        $\mathbf{x}$ = pop ($L$) ;
        if ($L$ was empty) then exit loop ;
    else
        bisect ($\mathbf{x}$), obtaining $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$;
        $\mathbf{x} = \mathbf{x}^{(1)}$ ;
        push ($L$, $\mathbf{x}^{(2)}$) ;
    end if ;
end loop
end IBP
```

The "rejection/reduction tests", mentioned in the algorithm are described in previous papers (specifically [13] and [14]), i.e.:

– switching between the componentwise Newton operator (for larger boxes) and Gauss-Seidel with inverse-midpoint preconditioner, for smaller ones,
– the sophisticated heuristic to choose the bisected component [13],
– an initial exclusion phase of the algorithm (deleting some regions, not containing solutions) – based on Sobol sequences [14].

Other possible variants (see, e.g., [11]) are not going to be considered.

3 Quadratic Approximations

3.1 Motivation

As stated above, the main tools used to narrow boxes in the branch-and-prune process are various forms of the Newton operator. This operator requires the computation of derivatives of (at least some of) the functions $f_i(\cdot)$. This computation – usually performed using the automatic differentiation process – is relatively costly. Because of that, in [14] the author proposed a heuristic using the Newton operator only for boxes that can be suspected to lie in the vicinity of a solution (or the solution manifold – in the underdetermined case). For other areas we try to base on 0th-order information only, i.e., function values and not gradients (or other derivatives). In [14] an "initial exclusion phase" was proposed, when regions are deleted using Sobol sequences, inner solution of the tolerance problem [19] and ε-inflation.

In [16] a similar approach (not using Sobol sequences, though) was considered for another problem – seeking Pareto sets of a multicriterion problem. Both papers show that this approach can improve the branch-and-bound type algorithms' efficiency dramatically – at least for some problems.

However, as for some areas the Newton operators do not perform well, we can try to use 2nd (or even higher) order information there. Obviously, this requires Hesse matrix computations, which is very costly, but can be worthwhile.

3.2 Quadratic Approximation

Each of the functions $f_i(x_1, \ldots, x_n)$ can be approximated by the 2nd order Taylor polynomial:

$$x \in \mathbf{x} \rightarrow f_i(x) \in \mathsf{f}_i(\check{x}) + \mathsf{g}(\check{x})^T \cdot (x - \check{x}) + \frac{1}{2} \cdot (x - \check{x})^T \cdot \mathsf{H}(\mathbf{x}) \cdot (x - \check{x}), \quad (1)$$

where $\mathsf{g}(\cdot)$ and $\mathsf{H}(\cdot)$ are interval extensions of the gradient and Hesse matrix of $f_i(\cdot)$, respectively.

By choosing a variable x_j, we can obtain a univariate formula: $x \in \mathbf{x} \rightarrow f_i(x) \in \mathbf{a}\boldsymbol{v}_j^2 + \mathbf{b} \cdot \boldsymbol{v}_j + \mathbf{c}$, where $v_j = x - \check{x}_j$.

Obviously:

$$\begin{aligned}
\mathbf{a} &= \frac{1}{2}\mathsf{H}_{jj}(\mathbf{x}),\\
\mathbf{b} &= \mathsf{g}_j(\check{x}) + \sum_{k \neq j} \mathsf{H}_{jk}(\mathbf{x}) \cdot (\mathbf{x}_k - \check{x}_k),\\
\mathbf{c} &= \frac{1}{2} \sum_{k \neq j} \Big(\mathsf{g}_k(\check{x}) \cdot (\mathbf{x}_k - \check{x}_k) + \mathsf{H}_{jk}(\mathbf{x}) \cdot (\mathbf{x}_k - \check{x}_k)^2 \Big)\\
&\quad + \sum_{k=1, k \neq j}^{n} \sum_{l=k+1}^{n} \mathsf{H}_{jk}(\mathbf{x}) \cdot (\mathbf{x}_k - \check{x}_k) \cdot (\mathbf{x}_l - \check{x}_l).
\end{aligned}$$

3.3 Interval Quadratic Equations

A quadratic equation is a well-known equation type of the form $ax^2 + bx + c = 0$. Methods of solving this equation in real numbers are common knowledge, nowadays.

How can such methods be generalized to the case when $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ are intervals and we have an interval $\mathbf{x}$ of possible values of the variable? Below, we present two solutions: a straightforward one and a more sophisticated one, based on the algorithm, described by Hansen [8].

The Straightforward Approach. This approach is a simple (yet not naïve) "intervalization" of the point-wise algorithm. It is provided by the author, but it also resembles techniques of [6].

Please note, it is assumed that the interval $\mathbf{a}$ is either strictly positive or strictly negative; for $a = 0$ the formulae for the quadratic equation do not make sense – even using extended interval arithmetic is of little help.

So, as for the non-interval case, we start with computing the discriminant of the equation: $\boldsymbol{\Delta} = \mathbf{b}^2 - 4\mathbf{ac}$. If all possible values of $\boldsymbol{\Delta}$ are guaranteed to be negative, i.e., $\overline{\Delta} < 0$ then for no quadratic approximation can there be any solutions, and we can discard the box $\mathbf{x}$. Otherwise, we set: $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta} \cap [0, +\infty]$ and compute the values:

$$\mathbf{x}^{(1)} = \frac{-\mathbf{b} - \sqrt{\boldsymbol{\Delta}}}{2\mathbf{a}}, \qquad \mathbf{x}^{(2)} = \frac{-\mathbf{b} + \sqrt{\boldsymbol{\Delta}}}{2\mathbf{a}}. \tag{2}$$

Please note, we cannot use the Viete formula $x^{(1)}x^{(2)} = \frac{c}{a}$ to compute one of the roots – $\mathbf{c}$ (and the other root) will often contain zero. However, we can use the Viete formula for narrowing:

$$\mathbf{x}^{(1)} \leftarrow \mathbf{x}^{(1)} \cap \frac{\mathbf{c}}{\mathbf{a}\mathbf{x}^{(2)}}, \qquad \mathbf{x}^{(2)} \leftarrow \mathbf{x}^{(2)} \cap \frac{\mathbf{c}}{\mathbf{a}\mathbf{x}^{(1)}}. \tag{3}$$

The two "interval solutions $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ can either be disjoint or not (if $0 \in \boldsymbol{\Delta}$ then they always coincide, but for strictly positive $\boldsymbol{\Delta}$ they can coincide, also). Now we can have the following possibilities:

- two disjoint intervals, both having nonempty intersections with $\mathbf{x}$ – the domain has been split, as for the Newton operator with extended arithmetic,
- two disjoint intervals, but only one of them has a nonempty intersection with $\mathbf{x}$ – then we can contract the domain; if the interval solution belongs to the interior of $\mathbf{x}$, we can prove the existence of a solution (as for the Newton operator),
- two coinciding intervals and at least one of them coincides with $\mathbf{x}$ – we narrow the domain, but cannot prove the existence or uniqueness of a solution,
- intervals disjoint with $\mathbf{x}$ – then we can discard this area, obviously.

Hansen's Approach. The book of Hansen and Walster [8] presents a sophisticated and efficient approach to solve quadratic equations with interval coefficients. This approach is applicable if $0 \in \mathbf{a}$, also.

The essence is to consider upper and lower functions of $\mathsf{f}(x) = [\underline{f}(x), \overline{f}(x)]$ and find x's, where $\underline{f}(x) \le 0 \le \overline{f}(x)$. It is assumed that $\overline{a} > 0$; if it is not, we can multiply both sides of the equation by (-1).

Please note, the upper and lower functions can be expressed as follows:

$$\underline{f}(x) = \begin{cases} \underline{a}x^2 + \underline{b}x + \underline{c} \text{ for } & x \ge 0 \\ \underline{a}x^2 + \overline{b}x + \underline{c} \text{ for } & x < 0 \end{cases}, \tag{4}$$

$$\overline{f}(x) = \begin{cases} \overline{a}x^2 + \overline{b}x + \overline{c} \text{ for } & x \ge 0 \\ \overline{a}x^2 + \underline{b}x + \overline{c} \text{ for } & x < 0 \end{cases}. \tag{5}$$

The condition $\overline{a} > 0$ implies that the upper function $\overline{f}(x)$ is convex. The lower function can be either convex, concave or neither convex nor concave.

The algorithm can be presented as follows:

```
initialize the list $S$ of points (represented by narrow intervals);
compute roots of $\underline{f}(x)$ and put them to the list $S$;
similarly, compute roots of $\overline{f}(x)$ and put them to the list $S$;
put $-\infty$ and/or $+\infty$ to $S$ if $\underline{f}(x) \le 0 \le \overline{f}(x)$ is fulfilled for these limits;
sort the list $L$ with respect to lower bounds of the entries;
if (there are no entries in $S$) then the equation has no solutions;
if (there are exactly two entries in $S$: $s_1$ and $s_2$) then
    the equation has one interval solution $[s_1, s_2]$;
if (there are exactly four entries in $S$: $s_1, \ldots, s_4$) then
    the equation has two interval solutions: $[s_1, s_2]$ and $[s_3, s_4]$;
if (there are exactly six entries in $S$: $s_1, \ldots, s_6$) then
    the equation has three interval solutions: $[s_1, s_2]$, $[s_3, s_4]$ and $[s_5, s_6]$;
```

In [8] it is specified that double roots should be stored twice on the list. Our implementation does not distinguish the cases when the discriminant $\Delta > 0$ or $\Delta = 0$, as it would be very difficult from the numerical point of view. So, the double root can be represented by two very close, probably coinciding, but different intervals.

Also, it is proven in the book that other numbers of solutions are not possible. The case of three interval solutions can occur when $\underline{f}(x)$ is nonconvex and it has four roots – two positive and two negative ones.

3.4 When to Use the Quadratic Approximation?

As stated above, crucial for designing successful interval algorithms is the choice of a proper heuristic to choose, arrange and parameterize interval tools for a specific box.

It seems reasonable to formulate the following advice *a priori*:

- not to use it when traditional (less costly) Newton operators, perform well,
- not to use it on too wide boxes – the ranges of approximations will grow at least in a quadratic range with the size,
- probably, also not to use it on too narrow boxes – higher order Taylor models do not have a better convergence than 1st order centered forms [18],
- if the Newton operator could not reduce one of the components as there were zeros in both the nominator and the denominator of the formula (see, e.g., [13]); this indicates that the box might contain a singular (or near-singular) point – we assume the Newton operator sets the flag `singular` in such a case.

In particular, the following heuristic appeared to perform well:

```
heuristic_for_use_of_quadratic_approximation
perform the Newton operator of some type on $\mathbf{x}$;
if ($\mathbf{x}$ was split or a solution has been verified) then return;
if (diameters of less than $m$ components of $\mathbf{x}$ do not exceed the value
      $\max\left(\frac{16.0}{n}, 1.0\right)$) then return; // the box is too large
if (diameters of more than $n - m$ components of $\mathbf{x}$ exceed the value
      $\frac{2.5}{2+n}$) then return; // the box is too small
if (some component of $\mathbf{x}$ has been narrowed on both sides) then return;
for ($k = 1$; $k \leq m$; ++$k$) do
      if (the $k$-th equation has at least one quadratic term) then
            compute the Hesse matrix of $f_k(\cdot)$ on $\mathbf{x}$;
            compute the function value and gradient at the midpoint of $\mathbf{x}$;
            for each variable, compute coefficients $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ of the quadratic
            approximation and try to solve the equation;
      end if
end for
end heuristic_for_use_of_quadratic_approximation
```

The above procedure does not take singularities into account. We can modify it, by changing the proper line to:

```
if (not singular and diameters of more than $n - m$ components of $\mathbf{x}$ exceed
      the value $\frac{2.5}{2+n}$) then return;
```

Both versions of the heuristic will be called: "basic" and "singularity checking" respectively, in the remainder.

4 Computational Experiments

Numerical experiments were performed on a computer with 16 cores, i.e., 8 Dual-Core AMD Opterons 8218 with 2.6 GHz clock. The machine ran under control of a Fedora 15 Linux operating system with the GCC 4.6.3, glibc 2.14 and the Linux kernel 2.6.43.8.

The solver is written in C++ and compiled using GCC compiler. The C-XSC library (version 2.5.3) [1] was used for interval computations. The parallelization (8 threads) was done with TBB 4.0, update 3 [2]. OpenBLAS 0.1 alpha 2.2 [3] was linked for BLAS operations.

According to previous experience (see [12]), 8 parallel threads were used to decrease computation time. Please note that parallelization does not affect the number of iterations, but the execution time only.

The following test problems were considered.

The first one is called the Hippopede problem [11,17] – two equations in three variables. Accuracy $\varepsilon = 10^{-7}$ was set.

The second problem, called Puma, arose in the inverse kinematics of a 3R robot and is one of typical benchmarks for nonlinear system solvers [4]. In the above form it is a well-determined (8 equations and 8 variables) problem with 16 solutions that are easily found by several solvers. To make it underdetermined the two last equations were dropped (as in [11]). The variant with 6 equations was considered in numerical experiments as the third test problem. Accuracy $\varepsilon = 0.05$ was set.

The third problem is well-determined – it is called Box3 [4] and has three equations in three variables. Accuracy ε was set to 10^{-5}.

The fourth one is a set of two equations – a quadratic one and a linear one – in five variables [7]. It is called the Academic problem. Accuracy $\varepsilon = 0.05$

The fifth problem is called the Brent problem – it is a well-determined algebraic problem, supposed to be "difficult" [5]. Presented results have been obtained for $N = 10$; accuracy was set to 10^{-7}.

And the last one is a well-determined one – the well-known Broyden-banded system [4,11]. In this paper we consider the case of $N = 16$. The accuracy $\varepsilon = 10^{-6}$ was set.

Results are given in Tables 1–4. The following notation is used in the tables:

- fun.evals, grad.evals, Hesse evals – numbers of functions evaluations, its gradients and Hesse matrices evaluations,
- bisecs – the number of boxes bisections,
- preconds – the number of preconditioning matrix computations (i.e., performed Gauss-Seidel steps),
- bis. Newt, del. Newt – numbers of boxes bisected/deleted by the Newton step,
- q.solv – the number of quadratic equations the algorithm was trying to solve,
- q.del.delta – the number of boxes deleted, because the discriminant of the quadratic equation was negative,
- q.del.disj. – the number of boxes deleted, because the solutions of a quadratic equation were disjoint with the original box,
- q.bisecs – the number of boxes bisected by the quadratic equations solving procedure,
- pos.boxes, verif.boxes – number of elements in the computed lists of boxes containing possible and verified solutions,
- Leb.pos., Leb.verif. – total Lebesgue measures of both sets,
- time – computation time in seconds.

5 Analysis of the Results

The proposed new version of our B&P algorithm resulted in an excellent speedup for the Brent problem (for one of the algorithm versions – for the Hippopede problem, also) and a significant one for Broyden16. For Puma6 the improvement was marginal and for problems Box3 and Academic, the new algorithm performed slightly worse than the version from [14]. It is worth noting that for the Academic problem, although the computation time was slightly worse, the accuracy was a bit better.

Table 1. Computational results for the algorithm version from [14]

Problem	Hippopede	Puma6	Box3	Academic	Brent10	Broyden16
fun. evals	440450	3620757	2387475	5568107	43399916	2894815943
grad.evals	502708	3162744	2278533	4776696	65109780	835991376
Hesse evals	—	—	—	—	—	—
bisections	115947	263181	379718	1193829	2822816	25765546
preconds	219599	447788	523947	2165486	2709154	8542793
bis. Newt.	13	99	27	92	432298	357371
del. Newt.	24209	53491	236390	208841	441141	18290280
pos.boxes	43210	184888	0	886722	473	0
verif.boxes	17069	2520	1	91	805	1
Leb.poss.	8e-18	3e-9	0.0	0.028	9e-82	0.0
Leb.verif.	0.003	3e-7	1e-25	1e-5	3e-69	4e-137
time (s)	<1	4	2	8	86	2752

Table 2. Computational results for the algorithm version using the quadratic approximation as described in Sect. 3

Problem	Hippopede	Puma6	Box3	Academic	Brent10	Broyden16
fun. evals	240659	3509119	2365265	5652078	43055720	2670617370
grad.evals	271652	3103980	2297800	4850533	64434484	787247696
Hesse evals	44	3786	32734	2547	7134	4135728
bisections	60353	257957	377474	1211743	2788715	24112367
preconds	119833	442924	523643	2197282	2672319	8736413
bis. Newt.	9	103	27	58	432276	358069
del. Newt.	14130	52539	235653	212840	439606	17048221
q.solv.	81	7480	65327	12691	19577	22768329
q.del.delta	0	0	0	0	0	9634
q.del.disj.	0	8	135	0	92	11924
q.bisecs	0	4	0	0	23	698
pos.boxes	21254	181240	0	898837	476	0
verif.boxes	9230	2424	1	88	803	1
Leb.poss.	4e-18	2e-9	0.0	0.027	9e-82	0.0
Leb.verif.	0.005	3e-7	1e-25	7e-6	3e-69	3e-137
time (s)	<1	4	2	7	87	2627

It seems, it is difficult to improve the performance of the algorithm, using the 2nd order information – yet possible, at least for some problems.

It is worth noting that the algorithm cannot improve the performance on bilinear problems – like the Rheinboldt problem, considered in the author's earlier papers (e.g., [11,13,14]). In such cases, we can try to transform the problem using symbolic techniques, e.g., the Gröbner basis theory (see, e.g., [15] and the references therein), but performance of this approach has yet to be investigated.

Table 3. Computational results for the algorithm version using the quadratic approximation solved by the Hansen method [8] and the basic heuristic

Problem	Hippopede	Puma6	Box3	Academic	Brent10	Broyden16
fun. evals	243336	3385431	2206661	5662094	42671592	2263804495
grad.evals	274676	2941828	2160304	4859064	63752945	691351646
Hesse evals	46	3586	29890	2822	5945	464702
bisections	60912	244445	355031	1213849	2757100	21219462
preconds	121182	417284	494074	2200090	2638126	5865559
bis. Newt.	10	111	28	46	429876	368942
del. Newt.	14174	52131	220406	212714	436998	14621392
q.solv.	87	7152	238676	14100	16410	2517097
q.del.delta	0	0	0	0	0	7915
q.del.disj.	0	8	105	0	91	7465
q.bisecs	1	4	0	0	83	11571
pos.boxes	21288	171496	0	900178	473	0
verif.boxes	9546	2384	1	100	804	1
Leb.poss.	4e-18	3e-9	0.0	0.027	9e-82	0.0
Leb.verif.	0.005	3e-7	1e-25	7e-6	3e-69	3e-137
time (s)	<1	4	2	7	84	2304

Table 4. Computational results for the algorithm version using the Hansen method [8] and the singularity checking version of the heuristic

Problem	Hippopede	Puma6	Box3	Academic	Brent10	Broyden16
fun. evals	556401	3314879	1959329	5612877	18681704	2251193535
grad.evals	635136	2927917	2036653	4870740	18875731	673122158
Hesse evals	1034	28890	539611	54090	571691	666926
bisections	150912	241173	249471	1203934	606847	20641684
preconds	277789	413004	411630	2179986	257325	5077374
bis. Newt.	7	111	26	43	307988	370729
del. Newt.	35268	50611	243524	2041267	318117	14409198
q.solv.	2063	57624	1079109	270431	1537925	3608072
q.del.delta	0	0	0	0	107	14095
q.del.disj.	0	144	107	0	20346	11195
q.bisecs	1	4	0	3	15847	13642
pos.boxes	65184	168944	0	897399	394	0
verif.boxes	11392	1920	1	99	811	1
Leb.poss.	5e-18	3e-9	0.0	0.027	2e-83	0.0
Leb.verif.	0.002	6e-9	1e-25	8e-6	2e-48	2e-144
time (s)	<1	4	3	7	31	2251

Performance of the method for the Hippopede problem is surprising – the algorithm version using the straightforward approach is the best there and the "singularity checking" heuristic version performs the worst. This remains to be carefully investigated in the future.

6 Conclusions

The proposed additional tool for interval branch-and-prune procedures, using quadratic approximations, allows us to improve the performance for some problems. The obtained improvement was minor for some cases, but significant, e.g., for the Broyden-banded problem and dramatic for the hard Brent problem. When the use of this tool is crucial, is going to be the subject of future research.

References

1. C-XSC interval library. http://www.xsc.de
2. Intel Threading Building Blocks. http://www.threadingbuildingblocks.org
3. OpenBLAS library. http://xianyi.github.com/OpenBLAS/
4. Non-polynomial nonlinear system benchmarks. https://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/node2.html
5. Difficult benchmark problems. http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/node6.html
6. Domes, F., Neumaier, A.: Constraint propagation on quadratic constraints. Constraints **15**(3), 404–429 (2010)
7. Goldsztejn, A., Jaulin, L.: Inner and outer approximations of existentially quantified equality constraints. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 198–212. Springer, Heidelberg (2006)
8. Hansen, E., Walster, W.: Global Optimization Using Interval Analysis. Marcel Dekker, New York (2004)
9. Kearfott, R.B.: Rigorous Global Search: Continuous Problems. Kluwer, Dordrecht (1996)
10. Kearfott, R.B., Nakao, M.T., Neumaier, A., Rump, S.M., Shary, S.P., van Hentenryck, P.: Standardized notation in interval analysis. http://www.mat.univie.ac.at/~neum/software/int/notation.ps.gz (2002)
11. Kubica, B.J.: Interval methods for solving underdetermined nonlinear equations systems. SCAN 2008 Proceedings. Reliable Comput. **15**(3), 207–217 (2011).
12. Kubica, B.J.: Shared-memory parallelization of an interval equations systems solver - comparison of tools. Pr. Nauk Politech. Warszawskiej. Elektron. **169**, 121–128 (2009)
13. Kubica, B.J.: Tuning the multithreaded interval method for solving underdetermined systems of nonlinear equations. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011, Part II. LNCS, vol. 7204, pp. 467–476. Springer, Heidelberg (2012)
14. Kubica, B. J.: Excluding regions using Sobol sequences in an interval branch-and-prune method for nonlinear systems. Presented ta SCAN2012 Conference, submitted to Reliable Computing.
15. Kubica, B.J., Malinowski, K.: An interval global optimization algorithm combining symbolic rewriting and componentwise Newton method applied to control a class of queueing systems. Reliable Comput. **11**(5), 393–411 (2005)
16. Kubica, B.J., Woźniak, A.: Tuning the interval algorithm for seeking Pareto sets of multi-criteria problems. In: Manninen, P., Öster, P. (eds.) PARA 2012. LNCS, vol. 7782, pp. 504–517. Springer, Heidelberg (2013)

17. Neumaier, A.: The enclosure of solutions of parameter-dependent systems of equations. In: Moore, R. (ed.) Reliability in Computing. Academic Press, San Diego (1988)
18. Neumaier, A.: Taylor forms - use and limits. Reliable Comput. **9**, 43–79 (2003)
19. Shary, S. P.: Finite-difference Interval Analysis. XYZ (2010) (in Russian)

The Definition of Interval-Valued Intuitionistic Fuzzy Sets in the Framework of Dempster-Shafer Theory

Ludmila Dymova(✉) and Pavel Sevastjanov

Institute of Computer and Information Science, Czestochowa University of Technology, Dabrowskiego 73, 42-200 Czestochowa, Poland
dymowa@icis.pcz.pl

Abstract. In this report, a critical analysis of conventional operations on interval-valued intuitionistic fuzzy values ($IVIFVs$) and their applicability to the solution of multiple criteria decision making ($MCDM$) problems in the interval-valued intuitionistic fuzzy setting are presented. It is shown that the classical definition of Atanassov's interval-valued intuitionistic fuzzy set (A-$IVIFS$) may lead to controversial results. Therefore, a new more constructive definition of A-$IVIFS$ is proposed. It is shown that this new definitions makes it possible to present $IVIFVs$ in the framework of interval-extended Dempster-Shafer theory of evidence (DST) as belief intervals with bounds presented by belief intervals.

Keywords: Interval-valued intuitionistic fuzzy values · Interval extended zero method · Interval-extended Dempster-Shafer theory

1 Introduction

Intuitionistic fuzzy set proposed by Atanassov [1], abbreviated here as A-IFS (the reasons for this are presented in [10]), is one of the possible generalizations of Fuzzy Sets Theory which currently is used mainly for solving multiple criteria decision making problems [7,13] and group decision making problem [15,16] when the values of local criteria (attributes) of alternatives and/or their weights are intuitionistic fuzzy values ($IFVs$). The concept of A-IFS is based on the simultaneous consideration of membership μ and non-membership ν of an element of a set to the set itself [1]. It is postulated that $0 \leq \mu + \nu \leq 1$.

Interval-Valued Intuitionistic Fuzzy Sets (A-$IVIFS$), were introduced in [2,3] as an interval extension of A-IFS. The fundamental characteristic of A-$IVIFS$ is that the values of its membership and non-membership functions are intervals rather than exact numbers. Atanassov [4,5] defined some operations and relations concerning A-$IVIFS$. Xu and Chen [22] proposed the interval-valued intuitionistic fuzzy weighted averaging operator for aggregation of interval-valued intuitionistic fuzzy values, and gave an application to $MCDM$ with interval-valued intuitionistic fuzzy information. In [20,21], the complete set of arithmetical operations on $IVIFVs$ (including the operations of $IVIFVs$ comparison) and based on them interval-valued intuitionistic fuzzy weighted averaging and weighted geometric operators was proposed.

Since different definitions of operations on $IFVs$ and their aggregation were proposed in the literature (see, for example [6,11]), in our recent paper [12] we analysed their merits and drawbacks and extracted those of them that provide the results of operations on $IFVs$ and aggregation with acceptable properties. As a result, a new set of operations on $IFVs$ in the framework of DST was proposed in [12].

As the classical A-IFS is an asymptotic case of A-$IVIFS$ when interval-valued membership and non-membership functions contract to points [3], we can expect that undesirable properties of classical operations on $IFVs$ revealed in [6,12] should be the same for operations on $IVIFVs$ presented in [20,21].

For these reasons the rest of this paper is set out as follows. Section 2 presents the basic definition of A-$IVIFS$, the commonly used arithmetical operations on $IVIFVs$, $IFVs$ and the methods for their comparison. The undesirable properties of these operations presented in [6,12] and some new ones revealed recently are analysed. In Sect. 3, we provide a critical analysis of commonly used basic definition of A-$IVIFS$ proposed in [3] to elicit its disadvantages which may lead to controversial results and propose a new more constructive definition of A-$IVIFS$. We show that this new definition makes it possible to present $IVIFVs$ in the framework of interval-extended DST as belief intervals with bounds presented by belief intervals. Finally, the concluding section summarises the paper.

2 The Basic Definitions of Interval-Valued Intuitionistic Fuzzy Set Theory

In [3], Atanassov and Gargov defined A-$IVIFS$ as follows.

Definition 1. *Let X be a finite universal set. Then an interval-valued intuitionistic fuzzy set $\tilde{A}$ in X is an object having the form*

$$\tilde{A} = \{\langle x, M_{\tilde{A}}(x), N_{\tilde{A}}(x)\rangle \,|x \in X\}, \tag{1}$$

where $M_{\tilde{A}}(x) \subset [0,1]$ and $N_{\tilde{A}}(x) \subset [0,1]$ are intervals such that for all $x \in X$

$$\sup M_{\tilde{A}}(x) + \sup N_{\tilde{A}}(x) \leq 1. \tag{2}$$

Hereinafter, we shall deal with $IVIFVs$. Therefore we shall use the following notation for $IVIFV$ A:

$$A = \left\langle [\mu_A^L, \mu_A^U],\, [\nu_A^L, \nu_A^U] \right\rangle, \tag{3}$$

where $[\mu_A^L, \mu_A^U]$ and $[\nu_A^L, \nu_A^U]$ are interval valued degrees of membership and non-membership to A.

The complete set of operations on $IVIFVs$ is presented in [20,21] as follows:

$$A \oplus B = \left\langle [\mu_A^L + \mu_B^L - \mu_A^L\mu_B^L,\, \mu_A^U + \mu_B^U - \mu_A^U\mu_B^U],\, [\nu_A^L\nu_B^L, \nu_A^U\nu_B^U] \right\rangle, \tag{4}$$

$$A \otimes B = \left\langle [\mu_A^L\mu_B^L,\, \mu_A^U\mu_B^U],\, [\nu_A^L + \nu_B^L - \nu_A^L\nu_B^L, \nu_A^U + \nu_B^U - \nu_A^U\nu_B^U] \right\rangle, \tag{5}$$

$$\lambda A = \left\langle [1 - (1 - \mu_A^L)^\lambda,\, 1 - (1 - \mu_A^U)^\lambda],\, [(\nu_A^L)^\lambda, (\nu_A^U)^\lambda] \right\rangle, \tag{6}$$

$$A^\lambda = \left\langle [(\mu_A^L)^\lambda,\, (\mu_A^U)^\lambda],\, [1 - (1 - \nu_A^L)^\lambda,\, 1 - (1 - \nu_A^U)^\lambda] \right\rangle, \tag{7}$$

where $\lambda > 0$.

Let $A_1, A_2, \ldots, A_n$ be $IVIFVs$ representing the values of n local criteria for some alternative and $w_1, w_2, \ldots, w_n$ be the real-valued weights of local criteria such that $w_i > 0$, $i = 1$ to n and $\sum\limits_{i=1}^{n} w_i = 1$.

Then based on the operations (4)–(7), the following interval-valued intuitionistic weighted arithmetic mean $IVIWAM$ and interval-valued intuitionistic weighted geometric mean $IVIWGM$ operators were obtained [20,21]:

$$IVIWAM(A_1, A_2, ..., A_n) =$$
$$\left\langle \left[1 - \prod_{i=1}^{n} (1 - \mu_{A_i}^L)^{w_i},\, 1 - \prod_{i=1}^{n} (1 - \mu_{A_i}^U)^{w_i}\right], \left[\prod_{i=1}^{n} (\nu_{A_i}^L)^{w_i},\, \prod_{i=1}^{n} (\nu_{A_i}^U)^{w_i}\right] \right\rangle, \tag{8}$$

$$IVIWGM(A_1, A_2, ..., A_n) =$$
$$\left\langle \left[\prod_{i=1}^{n} (\mu_{A_i}^L)^{w_i},\, \prod_{i=1}^{n} (\mu_{A_i}^U)^{w_i}\right], \left[1 - \prod_{i=1}^{n} (1 - \nu_{A_i}^L)^{w_i},\, 1 - \prod_{i=1}^{n} (1 - \nu_{A_i}^U)^{w_i}\right] \right\rangle. \tag{9}$$

To compare $IVIFVs$, the so-called score $S(A)$ and accuracy $H(A)$ functions were introduced in [20] as follows:

$$S(A) = \frac{\mu_A^L + \mu_A^U - \nu_A^L - \nu_A^U}{2},\ \ S(A) \in [-1, 1], \tag{10}$$

$$H(A) = \frac{\mu_A^L + \mu_A^U + \nu_A^L + \nu_A^U}{2},\ H(A) \in [0, 1]. \tag{11}$$

Then order relations between any pair of $IVIFVs$ A and B were presented in [20] as follows:

$$\begin{array}{l} If\ (S(A) > S(B)),\ then\, B\, is\, smaller\, than\, A; \\ If\ (S(A) = S(B)),\ then \\ (1)\, If\ (H(A) = H(B)),\ then\, A = B; \\ (2)\, If\ (H(A) < H(B))\, then\, A\, is\, smaller\, than\, B. \end{array} \tag{12}$$

It was proved in [20,21] that the operations (4)–(7) provide $IVIFVs$ and have the following algebraic properties:

Let A and B be $IVIFVs$. Then

$$A \oplus B = B \oplus A, \tag{13}$$

$$A \otimes B = B \otimes A, \tag{14}$$

$$\lambda(A \oplus B) = \lambda A \oplus \lambda B, \tag{15}$$

$$(A \otimes B)^{\lambda} = A^{\lambda} \otimes B^{\lambda}, \tag{16}$$

$$\lambda_1 A \oplus \lambda_2 A = (\lambda_1 + \lambda_2)A, \ \lambda_1, \lambda_2 > 0, \tag{17}$$

$$A^{\lambda_1} \otimes A^{\lambda_2} = A^{\lambda_1+\lambda_2}, \ \lambda_1, \lambda_2 > 0. \tag{18}$$

In the asymptotic case when $\mu_A^L \to \mu_A^U$ and $\nu_A^L \to \nu_A^U$, $IVIFV$ A reduces to the ordinary IFV a: $A \to a = \langle \mu_a, \nu_a \rangle$ and therefore from the above set of operations on $IVIFVs$ (4)–(12) we obtain the set of classical operations on ordinary $IFVs$:

$$a \oplus b = \langle \mu_a + \mu_b - \mu_a\mu_b, \nu_a\nu_b \rangle, \tag{19}$$

$$a \otimes b = \langle \mu_a\mu_b, \nu_a + \nu_b - \nu_a\nu_b \rangle. \tag{20}$$

$$\lambda a = \left\langle 1 - (1-\mu_a)^{\lambda}, \nu_a^{\lambda} \right\rangle, \tag{21}$$

$$a^{\lambda} = \left\langle \mu_a^{\lambda}, 1 - (1-\nu_a)^{\lambda} \right\rangle, \tag{22}$$

where $\lambda > 0$.

$$IWAM = w_1a_1 \oplus w_2a_2 \oplus ... \oplus w_na_n = \left\langle 1 - \prod_{i=1}^{n} (1-\mu_{a_i})^{w_i}, \prod_{i=1}^{n} \nu_{a_i}^{w_i} \right\rangle. \tag{23}$$

$$IWGM = a_1^{w_i} \otimes a_2^{w_i} ... \otimes a_n^{w_n} = \left\langle \prod_{i=1}^{n} \mu_{a_i}^{w_i}, 1 - \prod_{i=1}^{n} (1-\nu_{a_i})^{w_i} \right\rangle, \tag{24}$$

$$S(a) = \mu_a - \nu_a, \ \ S(a) \in [-1, 1], \tag{25}$$

$$H(a) = \mu_a + \nu_a, \ H(a) \in [0, 1]. \tag{26}$$

$$\begin{array}{l} If\ (S(a) > S(b)),\ then\, b\, is\, smaller\, than\, a; \\ If\ (S(a) = S(b)),\ then \\ (1)\, If\ (H(a) = H(b)),\ then\, a = b; \\ (2)\, If\ (H(a) < H(b))\ then\, a\, is\, smaller\, than\, b. \end{array} \tag{27}$$

It was proved in [19] that operations (19)–(22) provide $IFVs$ and have good algebraic properties (13)–(18). Nevertheless, in [12] we showed that operation (19)–(24) and (27) have some undesirable properties which may lead to the non-acceptable results in applications:

1. The addition (19) is not an addition invariant operation. Let a, b and c be $IFVs$. Then $a < b$ (according to (27)) does not always lead to $(a \oplus c) < (b \oplus c)$.
2. The operation (21) is not preserved under multiplication by a real-valued $\lambda > 0$, i.e., inequality $a < b$ (in sense of (27)) does not necessarily imply $\lambda a < \lambda b$.
3. An important problem with the aggregation operation (23) is that it is not consistent with the aggregation operation on the ordinary fuzzy sets (when $\mu = 1 - \nu$). This can be easily seen from the following example.

Example 1. Let $A = \langle 0.1, 0.9\rangle$, $B = \langle 0.9, 0.1\rangle$ and $w_1 = w_2 = 0.5$. It is easy to see that A and B are IF representations of ordinary fuzzy numbers. Then in the framework of ordinary fuzzy sets, we get Ordinary Weighted Arithmetic Mean $OWAM = w_1\mu_A + w_2\mu_B = 0.5 \cdot 0.1 + 0.5 \cdot 0.9 = 0.5$ and in the framework of A-IFS from (23), we obtain $IWAM = \langle 0.7, 0.3\rangle$. We can see that the resulting value of μ obtained using $IWAM$ is considerably greater than that obtained from $OWAM$.

4. Another problem with the aggregation operation (23) is that it is not monotone with respect to the ordering (27). Let a, b and c be $IFVs$. Then $b > c > a$ (in sense of (27)) does not always lead to $IWAM(b,a) > IWAM(c,a)$.
   Recently we found some undesirable properties of operations (20) and (24):
5. The multiplication (20) is not always monotone with respect to the ordering (27). Consider an example:

Example 2. Let $a = \langle 0.4, 0.5\rangle$, $b = \langle 0.35, 0.44\rangle$, $c = \langle 0.001, 0.999\rangle$. Then $S(a) = -0.1, S(b) = -0.09$ and therefore according to (27) we get $b > a$. On the other hand, $a \otimes c = \langle 0.401, 0.9995\rangle$, $b \otimes c = \langle 0.351, 0.99944\rangle$, $S(a \otimes c) = -0.5985$, $S(b \otimes c) = -0.64844$. Therefore $S(a \otimes c) > S(b \otimes c)$ and $a \otimes c > b \otimes c$ opposite to $b > a$.

6. The aggregation operation (24) is not monotone with respect to the ordering (27). Consider an example: Let $a = \langle 0.4, 0.5\rangle$, $b = \langle 0.35, 0.44\rangle$, $c = \langle 0.1, 0.9\rangle$, $w_1 = 0.3, w_2 = 0.7$. Since $S(a) = -0.1$ and $S(b) = -0.09$ we get $b > a$. hand, since $IWGM(a,c) = \langle 0.15155, 0.83795\rangle$, $IWGM(b,c) = \langle 0.1456, 0.83235\rangle$, $S(IWGM(a,c)) = -0.6864$ and $S(IWGM(a,b)) = -0.6868$ we have $IWGM(a,c) > IWGM(b,c)$ opposite to the $b > a$.

As the classical A-IFS is the asymptotic case of A-$IVIFS$ when interval-valued membership and non-membership functions contract to points [3], we can expect that undesirable properties of classical operations on $IFVs$ presented above should be the same for operations on $IVIFVs$ from [20,21] presented in this section.

3 The New Definitions of *IVIFV*

3.1 A New Definition of *IVIFV* in the Framework of *A-IFS*

First of all we shall analyse the commonly used definition of A-$IVIFS$ (and $IVIFV$ as well), proposed by Atanassov and Gargov [3] (see Definition 1 above) and reveal its drawbacks. As a result, a new more constructive definition will be proposed. It is implicitly assumed in Definition 1 that the upper bound of $IVIFV\ A = \langle [\mu_A^L, \mu_A^U], [\nu_A^L, \nu_A^U] \rangle$, is an ordinary IFV, i.e, $\mu_A^U + \nu_A^U \leq 1$. If so, there will be a hesitation degree π_A^U at the upper bound of A such that $\mu_A^U + \nu_A^U + \pi_A^U = 1$. It is easy to see that in that case all other possible combinations of μ_A, ν_A, π_A in the intervals $[\mu_A^L, \mu_A^U]$, $[\nu_A^L, \nu_A^U]$, $[\pi_A^L, \pi_A^U]$ provide $\mu_A + \nu_A + \pi_A < 1$. Therefore, there is only one IFV at the upper bound of $IVIFV\ A$. This a first obvious drawback of Definition 1.

Let us consider $A = \langle [0.6, 0.7], [0.1, 0.25] \rangle$. Since $\mu_A^U + \nu_A^U = 0.7 + 0.25 = 0.95 < 1$, then according to the Definition 1, A is a correct $IVIFV$. On the other hand, on the bounds of A we have the following values of score functions $S(A)^U = \mu_A^U - \nu_A^U = 0.45, S(A)^L = \mu_A^L - \nu_A^L = 0.5$. Since $S(A)^L > S(A)^U$, then according to the rule (27) the lower bound of considered $IVIFVA$ is greater that its upper bound. Obviously, such result have no reasonable explanations.

It is easy to see that the Definition 1 says nothing about the left bound of $IVIFV$. To overcome this problem, currently the following formal approach is used (see, for example [18,23]):

If A is $IVIFV$ then $A = \langle [\mu_A^L, \mu_A^U], [\nu_A^L, \nu_A^U], [\pi_A^L, \pi_A^U] \rangle$, where $\pi_A^U = 1 - \mu_A^L - \nu_A^L$, $\pi_A^L = 1 - \mu_A^U - \nu_A^U$. It is easy to show that such approach provides wrong results. Let us consider $A = \langle [0.3, 0.6], [0.1, 0.2] \rangle$. Since $\mu_A^U + \nu_A^U = 0.6 + 0.2 = 0.8 < 1$, then according to the Definition 1, A is a correct $IVIFV$. On the other hand $\pi_A^U = 1 - \mu_A^L - \nu_A^L = 1 - 0.3 - 0.1 = 0.6$ and $\pi_A^L = 1 - \mu_A^U - \nu_A^U = 1 - 0.6 - 0.2 = 0.2$. Then $\mu_A^U + \nu_A^U + \pi_A^U = 1.4$. Therefore the upper bound of $IVIFV$ A is not a correct IFV. In our opinion, the above problem is based on faulty treatment of interval arithmetic rules. The expressions for calculation of π_A^U and π_A^L are implicitly based on the assumption that $[\mu_A^L, \mu_A^U] + [\nu_A^L, \nu_A^U] + [\pi_A^L, \pi_A^U] = 1$. Since in the left hand side of this expression we have an interval and in the right hand side we have a real value, then an equality is impossible. Taking into account the above analysis, here we introduce a new definition of $IVIFV$ such that the upper and lower bound of $IVIFV$ will always be ordinary $IFVs$ and upper bound will always be greater than a lower one in sense of rule (27).

Definition 2. *Let X be a finite universal set. Then an interval-valued intuitionistic fuzzy set $\tilde{A}$ in X is an object having the form*

$$\tilde{A} = \{ \langle x, M_{\tilde{A}}(x), N_{\tilde{A}}(x) \rangle \, | x \in X \}, \tag{28}$$

where $M_{\tilde{A}}(x) \subset [0, 1]$ and $N_{\tilde{A}}(x) \subset [0, 1]$ are intervals such that for all $x \in X$

$$\sup M_{\tilde{A}}(x) + \inf N_{\tilde{A}}(x) \leq 1, \inf M_{\tilde{A}}(x) + \sup N_{\tilde{A}}(x) \leq 1. \tag{29}$$

The definition for $IVIFV$ is obtained from Definition 2 as follows.

Definition 3. *An interval-valued intuitionistic fuzzy value A is an object having the form*

$$A = \langle [\mu_A^L, \mu_A^U], [\nu_A^L, \nu_A^U] \rangle, \tag{30}$$

where μ_A^L, μ_A^U, ν_A^L, $\nu_A^L \in [0,1]$ *and*

$$\mu_A^L + \nu_A^U \leq 1, \; \mu_A^U + \nu_A^L \leq 1. \tag{31}$$

It is easy to see that $\langle \mu_A^U, \nu_A^L \rangle$ *is the maximal (in sense of rule (27)) IFV attainable in* $A = \langle [\mu_A^L, \mu_A^U], [\nu_A^L, \nu_A^U] \rangle$. *Therefore* $\langle \mu_A^U, \nu_A^L \rangle$ *is the upper bound of IVIFV. Similarly,* $\langle \mu_A^L, \nu_A^U \rangle$ *is the lower bound of IVIFV A. It is easy to see that the upper bound of such IVIFV is always not lesser of its lower bound in sense of rule (27) and these bounds are correct IFVs. It is clear that for any correct IFV* $\langle \mu, \nu \rangle$ *such that* $\mu \in [\mu_A^L, \mu_A^U]$ *and* $\nu \in [\nu_A^L, \nu_A^U]$ *we have* $\langle \mu_A^L, \nu_A^U \rangle \leq \langle \mu, \nu \rangle \leq \langle \mu_A^U, \nu_A^L \rangle$.

So it is implicitly assumed that $IVIFV$ is a set of correct $IFVs$ bounded according to the Definition 3. This is in compliance with a practice of obtaining $IVIFVs$ in many real-world situations and seems to be justified enough from methodological point of view. Really, when we are dealing with intervals, they should have the left bounds that are lesser than right ones. If we analyse real-valued intervals, then they should consist of real values. If we introduce integer-valued intervals, then they should consist of integer numbers. Similarly, $IVIFVs$ should be intervals consist of only $IFVs$.

3.2 A New Definition of *IVIFV* in the Framework of Interval-Extended *DST*

At first, we present a brief description of some fundamentals of DST needed for the subsequent analysis.

The origins of the Dempster-Shafer theory go back to the work by A.P. Dempster [8,9] who developed a system of upper and lower probabilities. Following this work his student G. Shafer [14] included in his 1976 book "A Mathematical Theory of Evidence" a more thorough explanation of belief functions.

Below we provide an introduction to basic ideas of this theory. Assume A is a subset of X. It is important to note that a subset A may be treated also as a question or proposition and X as a set of propositions or mutually exclusive hypotheses or answers [17]. A DST belief structure has associated with it a mapping m, called basic assignment function (or mass assignment function), from subsets of X into a unit interval, $m : 2^X \to [0,1]$ such that $m(\emptyset) = 0$, $\sum\limits_{A \subseteq X} m(A) = 1$. The subsets of X for which the mapping does not assume a zero value are called focal elements.

In the framework of classical Dempster-Shafer approach, it is assumed that the null set is never a focal element. In [14], Shafer introduced a number of measures associated with DST belief structure. The measure of belief is a mapping $Bel : 2^X \to [0,1]$ such that for any subset B of X

$$Bel(B) = \sum_{\emptyset \neq A \subseteq B} m(A). \tag{32}$$

It is shown in [14] that m can be uniquely recovered from *Bel*. A second measure introduced by Shafer [14] is a measure of plausibility. The measure of plausibility associated with m is a mapping $Pl\colon 2^X \to [0,1]$ such that for any subset B of X

$$Pl(B) = \sum_{A \cap B \neq \emptyset} m(A). \tag{33}$$

It is easy to see that $Bel(B) \leq Pl(B)$. DST provides an explicit measure of ignorance about an event B and its complementary $\overline{B}$ as a length of an interval [*Bel(B),Pl(B)*] called the belief interval (BI). It can also be interpreted as an interval enclosing the "true probability" of B [14].

The above information of DST is quite enough for the presentation of A-IFS in terms of DST. Firstly, we show that in the framework of DST the triplet $\mu_A(x)$, $\nu_A(x)$, $\pi_A(x)$ represents the basic assignment function. Really, when analysing any situation in context of A-IFS, we implicitly deal with the following three hypotheses: $x \in A$, $x \notin A$ and the situation when both the hypotheses $x \in A$, $x \notin A$ can not be rejected (the case of hesitation). In the spirit of DST, we can denote these hypotheses as Yes ($x \in A$), No ($x \notin A$) and (Yes, No) (the case of hesitation when both the hypotheses $x \in A$ and $x \notin A$ can not be rejected).

In this context, $\mu_A(x)$ may be treated as the probability or evidence of $x \in A$, i.e., as the focal element of the basic assignment function: $m(Yes) = \mu_A(x)$. Similarly, we can assume that $m(No) = \nu_A(x)$. Since $\pi_A(x)$ is usually treated as a hesitation degree, a natural assumption is $m(Yes, No) = \pi_A(x)$. Taking into account that $\mu_A(x)+\nu_A(x)+\pi_A(x) = 1$ we come to the conclusion that the triplet $\mu_A(x)$, $\nu_A(x)$, $\pi_A(x)$ represents a correct basic assignment function. According to the DST formalism we get $Bel_A(x) = m(Yes) = \mu_A(x)$ and $Pl_A(x) = m(Yes) + m(Yes, No) = \mu_A(x) + \pi_A(x) = 1 - \nu_A(x)$.

Therefore $IFVA(x) = \langle \mu_A(x), \nu_A(x) \rangle$ may be represented as follows: $A(x) = BI_A(x) = [Bel_A(x), Pl_A(x)] = [\mu_A(x), 1 - \nu_A(x)]$ (see [11,12] for more detailed and formal definitions).

At first glance, this definition seems to be a simple redefinition of A-IFS in terms of Interval Valued Fuzzy Sets, but in [11,12] we showed that using DST semantics it is possible to enhance the performance of A-IFS when dealing with the operations on $IFVs$ and $MCDM$ problems. Based on the described above link between A-IFS and DST, the Definition 2 can be rewritten as follows:

Definition 4. *Let X be a finite universal set. Then an interval-valued intuitionistic fuzzy set $\tilde{A}$ in X is an object having the form*

$$\tilde{A} = \left\{ \langle x, \, [BI_{\tilde{A}}(x)] \rangle \, | x \in X \right\}, \tag{34}$$

where

$$[BI_{\tilde{A}}(x)] = \left[BI^L_{\tilde{A}}(x), BI^U_{\tilde{A}}(x) \right] \tag{35}$$

is the belief interval with bounds presented by the belief intervals:

$$BI^L_{\tilde{A}}(x) = \left[\inf M_{\tilde{A}}(x), 1 - \sup N_{\tilde{A}}(x) \right], BI^U_{\tilde{A}}(x) = \left[\sup M_{\tilde{A}}(x), 1 - \inf N_{\tilde{A}}(x) \right], \tag{36}$$

where $M_{\tilde{A}}(x) \subset [0,1]$ *and* $N_{\tilde{A}}(x) \subset [0,1]$ *are intervals such that for all* $x \in X$

$$\sup M_{\tilde{A}}(x) + \inf N_{\tilde{A}}(x) \leq 1, \inf M_{\tilde{A}}(x) + \sup N_{\tilde{A}}(x) \leq 1. \tag{37}$$

The definition for $IVIFV$ is obtained from Definition 4 as follows.

Definition 5. *An interval-valued intuitionistic fuzzy value A is an object having the form*

$$A = [BI_A] = \left[BI_A^L, BI_A^U\right], \tag{38}$$

where

$$BI_A^L = \left[Bel_A^L,\, Pl_A^L\right] = \left[\mu_A^L, 1-\nu_A^U\right],\, BI_A^U = \left[Bel_A^U,\, Pl_A^U\right] = \left[\mu_A^U, 1-\nu_A^L\right], \tag{39}$$

$\mu_A^L,\, \mu_A^U,\, \nu_A^L,\, \nu_A^L \in\ [0,1]$ *and*

$$\mu_A^L + \nu_A^U \leq 1,\, \mu_A^U + \nu_A^L \leq 1. \tag{40}$$

It is important that according to the last definition all intervals $[BI] = \left[BI^L, BI^U\right]$ with bounds presented by correct belief intervals such that $BI^U \geq BI^L$ may be treated as $IVIFVs$. Since in the above definitions we have introduced belief intervals bounded by belief intervals ($BIBBI$), we can say that in our case, we are dealing with the someway interval-extended version of DST. We do not intend to develop here the whole interval-extended DST and restrict ourselves only by consideration of $BIBBI$.

Since in [12] we obtained the complete set of operations on $IFVs$ in the framework of DST, which are free of the drawbacks of classical operation laws of A-IFS, our future work will be focused on the obtaining the set of operations on $IVIFVs$ in the framework of interval-extended DST which is free of the described above drawbacks of the classical approach.

4 Conclusion

This report is devoted to the critical analysis of the commonly used definition of interval-valued intuitionistic fuzzy sets (A-$IVIFS$) and the operations on the interval-valued intuitionistic fuzzy values ($IVIFVs$). It is shown that this definition leads to the controversial results and the classical operations on $IVIFV$ have some undesirable properties.

Therefore, a new more constructive definition of A-$IVIFS$ is proposed. It is shown that this new definition makes it possible to present $IVIFVs$ in the framework of interval-extended Dempster-Shafer theory of evidence (DST) as belief intervals with bounds presented by belief intervals. In [12], we obtained the complete set of operations on $IFVs$ in the framework of DST, which are free of the drawbacks of classical operation laws of A-IFS. Therefore, our future work will be focused on the obtaining the set of operations on $IVIFVs$ in the framework of interval-extended DST which is free of the drawbacks of classical approach.

References

1. Atanassov, K.T.: Intuitionistic fuzzy sets. Fuzzy Sets Syst. **20**, 87–96 (1986)
2. Atanassov, K.: Review and new results on intuitionistic fuzzy sets. Preprint IM-MFAIS-1-88, Sofia (1988)
3. Atanassov, K.T., Gargov, G.: Interval valued intuitionistic fuzzy sets. Fuzzy Sets Syst. **31**, 343–349 (1989)
4. Atanassov, K.: New operations defined over the intuitionistic fuzzy sets. Fuzzy Sets Syst. **61**, 137–142 (1994)
5. Atanassov, K.: Operators over interval-valued intuitionistic fuzzy sets. Fuzzy Sets Syst. **64**, 159–174 (1994)
6. Beliakov, G., Bustince, H., Goswami, D.P., Mukherjee, U.K., Pal, N.R.: On averaging operators for atanassov's intuitionistic fuzzy sets. Inf. Sci. **182**, 1116–1124 (2011)
7. Chen, S.M., Tan, J.M.: Handling multicriteria fuzzy decision-making problems based on vague set theory. Fuzzy Sets Syst. **67**, 163–172 (1994)
8. Dempster, A.P.: Upper and lower probabilities induced by a multi-valued mapping. Ann. Math. Stat. **38**, 325–339 (1967)
9. Dempster, A.P.: A generalization of bayesian inference (with discussion). J. Royal Stat. Soc. Ser. B **30**(2), 208–247 (1968)
10. Dubois, D., Gottwald, S., Hajek, P., Kacprzyk, J., Prade, H.: Terminological difficulties in fuzzy set theory-the case of "intuitionistic fuzzy sets". Fuzzy Sets Syst. **156**, 485–491 (2005)
11. Dymova, L., Sevastjanov, P.: An interpretation of intuitionistic fuzzy sets in terms of evidence theory. decision making aspect. Knowl.-Based Syst. **23**, 772–782 (2010)
12. Dymova, L., Sevastjanov, P.: The operations on intuitionistic fuzzy values in the framework of dempster-shafer theory. Knowl.-Based Syst. **35**, 132–143 (2012)
13. Hong, D.H., Choi, C.-H.: Multicriteria fuzzy decision-making problems based on vague set theory. Fuzzy Sets Syst. **114**, 103–113 (2000)
14. Shafer, G.: A Mathematical Theory of Evidence. Princeton University Press, Princeton (1976)
15. Szmidt, E., Kacprzyk, J.: Intuitionistic fuzzy sets in decision making. Notes IFS **2**, 15–32 (1996)
16. Szmidt, E., Kacprzyk, J.: Remarks on some applications on intuitionistic fuzzy sets in decision making. Notes IFS **2**, 22–31 (1996)
17. Vasseur, P., Pegard, C., Mouad, E., Delahoche, L.: Perceptual organization approach based on dempster-shafer theory. Pattern Recogn. **32**, 1449–1462 (1999)
18. Wang, Z., Li, K.W., Xu, J.: A mathematical programming approach to multi-attribute decision making with interval-valued intuitionistic fuzzy assessment information. Expert Syst. Appl. **38**, 12462–12469 (2011)
19. Xu, Z.: Intuitionistic preference relations and their application in group decision making. Inf. Sci. **177**, 2363–2379 (2007)
20. Xu, Z.: Intuitionistic fuzzy aggregation operators. IEEE Trans. Fuzzy Syst. **15**, 1179–1187 (2007)
21. Xu, Z.: Methods for aggregating interval-valued intuitionistic fuzzy information and their application to decision making. Control and Decision **22**, 215–219 (2007)
22. Xu, Z., Chen, J.: Approach to group decision making based on interval-valued intuitionistic judgement matrices. Syst. Eng. Theory Pract. **27**, 126–133 (2007)
23. Yue, Z.: An approach to aggregating interval numbers into interval-valued intuitionistic fuzzy information for group decision making. Expert Syst. Appl. **38**, 6333–6338 (2011)

Interval Finite Difference Method for Solving the Problem of Bioheat Transfer Between Blood Vessel and Tissue

Malgorzata A. Jankowska(✉)

Institute of Applied Mechanics, Poznan University of Technology,
Jana Pawla II 24, 60-965 Poznan, Poland
malgorzata.jankowska@put.poznan.pl

Abstract. The paper concerns a problem of bioheat transfer between a single large blood vessel and a surrounding tissue. On the basis of the conventional finite difference scheme with the appropriate truncation error terms included, the interval finite difference method is proposed. The interval values that contain the local truncation error of the conventional scheme can be approximated in the way described.

Keywords: Interval finite difference method · Interval arithmetic · Vessel model of bioheat transfer · Uncertain values of parameters

1 Introduction

Studies of heat transfer in biological systems are still intensively carried out. Unquestionably, the most significant system for distribution of heat throughout the body is the cardiovascular system with blood that transport heat. One of many problems discussed in this research area is the one that concerns the heat transfer in the perfused tissue surrounding a single blood vessel of a large diameter (see, e.g. Chato 1980; Huang et al. 1994; Majchrzak and Mochnacki 1999; 2001; Majchrzak 2011). For such an issue the mathematical models can be proposed and then the problems solved with some computational method. As we know, conventional numerical methods usually used for solving initial-boundary value problems with governing equations given by ordinary or partial differential equations (ODE, PDE), do not allow for taking into account the uncertain values of parameters present in a problem formulation. Moreover, the errors of the conventional methods are also neglected. The objective of the paper is to propose an interval finite difference scheme for solving the bioheat transfer problem considered. Such interval method provides interval solutions that include the exact solution of the problem. Its computer implementation in the floating-point interval arithmetic (see, e.g. Jankowska 2010), together with the representation of the initial data in the form of machine intervals, allows to obtain interval solutions that contain all possible numerical errors. Note that such interval approaches for solving some initial-boundary value problems were also used in e.g. (Jankowska and Sypniewska-Kaminska 2012, 2013) and (Marciniak and Szyszka 2013).

2 The Bioheat Transfer Problem

Now we provide a mathematical formulation of the model that concerns the bioheat transfer in the perfused tissue in the presence of a single large blood vessel (see, e.g. Chato 1980; Huang et al. 1994; Majchrzak and Mochnacki 1999; 2001; Majchrzak 2011). The initial value problem that describes the change of the blood temperature along the vessel axis is coupled with the boundary value problem that concerns the temperature distribution in the tissue. Both problems are given in an original dimensional form, as in e.g. (Majchrzak 2011) and then in the non-dimensional one. Finally, we submit some general form for which the conventional and interval finite difference schemes are presented in Sect. 3.

Following, e.g. (Huang et al. 1994; Majchrzak 2011), we first consider the ordinary differential equation that describes the temperature distribution of blood flowing with a given velocity v_{B} in a vessel of length L (see Fig. 1). The cross-section of the blood vessel is assumed to be a circle of a radius equal to R_1. The temperature of blood at entrance is $w_{\mathrm{B}0}$ and it serves as an initial condition for the governing equation. We have

$$\begin{gathered}\frac{dw_{\mathrm{B}}}{dx}(x) = \frac{2\alpha}{c_{\mathrm{B}}\rho_{\mathrm{B}}v_{\mathrm{B}}R_1}\left(w_w(x) - w_{\mathrm{B}}(x)\right) + \frac{\dot{Q}_{\mathrm{B}met}}{c_{\mathrm{B}}\rho_{\mathrm{B}}v_{\mathrm{B}}}, \quad 0 < x < L, \\ w_{\mathrm{B}}(0) = w_{\mathrm{B}0},\end{gathered} \tag{1}$$

where $w_{\mathrm{B}} = w_{\mathrm{B}}(x)$ denotes the average temperature of blood in the vessel, x is a spatial coordinate that describes the position in the vessel, $w_w = w_w(x)$ is the temperature of the vessel wall, α [W/(m^2·K)] is the coefficient of convection heat transfer between blood and tissue, c_{B} [J/(kg·K)] is the specific heat of blood, ρ_{B} [kg/m^3] is the mass density of blood, v_{B} [m/s] is the velocity of blood, $\dot{Q}_{\mathrm{B}met}$ [W/m^3] is the metabolic volumetric heat generation rate in the vessel. Note that a unit of measurement for the temperature is Kelvin [K] and for the length is meter [m]. Furthermore, the unknown values of the temperature in (1) are $w_{\mathrm{B}} = w_{\mathrm{B}}(x)$ and $w_w = w_w(x)$, respectively.

On the other hand, the temperature distribution in the tissue surrounding a single blood vessel can be described by the boundary value problem of the form

$$\begin{gathered}\lambda\frac{d^2w}{dr^2}(r) + \frac{\lambda}{r}\frac{dw}{dr}(r) + c_{\mathrm{B}}\rho_{\mathrm{B}}G_{\mathrm{B}}\left(w_a - w(r)\right) + \dot{Q}_{met} = 0, \quad R_1 < r < R_2, \\ \lambda\frac{dw}{dr}(R_1) = \alpha\left(w(R_1)|_x - w_{\mathrm{B}}(x)\right), \quad w(R_2) = w_t,\end{gathered} \tag{2}$$

where $w = w(r)$ denotes the temperature in the tissue, r is a radial coordinate that describes the distance from a vessel axis, λ [W/(m·K)] is the thermal conductivity of the tissue, G_{B} [1/s] is the blood perfusion coefficient, w_a is the temperature of blood in aorta, w_t is the temperature of the tissue at the position $r = R_2$, $\dot{Q}_{met}$ [W/m^3] is the metabolic volumetric heat generation rate in the tissue. Note that $\lambda = \kappa c\rho$, where κ [m^2/s] denotes the thermal diffusivity of the tissue and c, ρ – the specific heat and the mass density of the tissue, respectively. Moreover, $G_{\mathrm{B}} = W_{\mathrm{B}}/\rho_{\mathrm{B}}$, where W_{B} [kg/(m^3·s)] denotes the blood

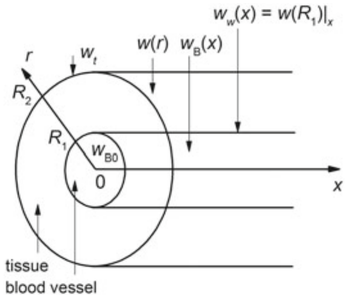


Fig. 1. A diagram of a single blood vessel surrounded by the tissue.

perfusion rate per unit volume of the tissue. The unknown value of the temperature in (2) is $w = w(r)$. The governing equations in (1), (2) are coupled and we have $w_w(x) = w(R_1)|_x$. Moreover, we use the notation $w = w(r)|_x$ to underline that we consider the temperature of the tissue in the position x along the vessel axis.

Subsequently, we make use of two kinds of vessel models as proposed in e.g. (Huang et al. 1994). In the first case (a traversing vessel model), a large blood vessel merely traverses through the perfused tissue. Hence, the temperature w_a of the blood in aorta is assumed to remain constant. We take $w_a = w_{\mathrm{B}0}$. Then, we analyze a supplying vessel model. It assumes that the temperature w_a of the blood in aorta is equal to the bulk temperature of the blood flowing in the vessel (also called a supplying vessel). Hence, we have $w_a = w_{\mathrm{B}}(x)$. Now we transform the Eqs. (1), (2) to the non-dimensional form. We define new coordinates in the following way

$$\begin{gathered} u_{\mathrm{B}}(\xi) = (w_{\mathrm{B}}(x) - w_{\mathrm{ref}})/w_{\mathrm{ref}}, \quad u(\rho) = (w(r) - w_{\mathrm{ref}})/w_{\mathrm{ref}}, \\ \xi = x/L, \quad \rho = r/R_2, \end{gathered} \tag{3}$$

where w_{ref} denotes some reference value of the temperature. Hence, for the initial value problem (1) we obtain

$$\begin{gathered} \frac{du_{\mathrm{B}}}{d\xi}(\xi) = g_{1\mathrm{B}}\left(u_w(\xi) - u_{\mathrm{B}}(\xi)\right) + g_{2\mathrm{B}}, \quad 0 < \xi < 1, \\ u_{\mathrm{B}}(0) = u_{\mathrm{B}0}, \end{gathered} \tag{4}$$

where

$$u_{\mathrm{B}0} = \frac{w_{\mathrm{B}0} - w_{\mathrm{ref}}}{w_{\mathrm{ref}}}, \quad g_{1\mathrm{B}} = \frac{2\alpha L}{c_{\mathrm{B}}\rho_{\mathrm{B}}v_{\mathrm{B}}R_1}, \quad g_{2\mathrm{B}} = \frac{\dot{Q}_{\mathrm{B}met}L}{c_{\mathrm{B}}\rho_{\mathrm{B}}v_{\mathrm{B}}w_{\mathrm{ref}}}. \tag{5}$$

Then, for the boundary value problem (2) we get

$$\begin{gathered} \frac{d^2u}{d\rho^2}(\rho) + \frac{1}{\rho}\frac{du}{d\rho}(\rho) + g_1\left(u_a - u(\rho)\right) + g_2 = 0, \quad \rho_{\min} < \rho < 1, \\ \frac{du}{d\rho}(\rho_{\min}) = g_3\left(u(\rho_{\min})|_{\xi} - u_{\mathrm{B}}(\xi)\right), \quad u(1) = u_t, \end{gathered} \tag{6}$$

where

$$g_1 = \frac{R_2^2 c_{\mathrm{B}} \rho_{\mathrm{B}} G_{\mathrm{B}}}{\lambda}, \quad g_2 = \frac{R_2^2 \dot{Q}_{met}}{\lambda w_{\mathrm{ref}}} = \widetilde{\dot{Q}}_{met} = \mathrm{Os}, \quad g_3 = \frac{\alpha R_2}{\lambda} = \mathrm{Bi}, \\ u_t = \frac{w_t - w_{\mathrm{ref}}}{w_{\mathrm{ref}}}, \quad \rho_{\min} = \frac{R_1}{R_2}, \tag{7}$$

and Os denotes the Ostrogradsky number and Bi – the Biot number. Finally, if values of the material parameters of the tissue, i.e. κ, c, ρ, are known, then the non-dimensional parameter g_1 can be determined with another formula

$$g_1 = \widetilde{c}_{\mathrm{B}} \widetilde{\rho}_{\mathrm{B}} \widetilde{G}_{\mathrm{B}}, \quad \widetilde{c}_{\mathrm{B}} = c_{\mathrm{B}}/c, \quad \widetilde{\rho}_{\mathrm{B}} = \rho_{\mathrm{B}}/\rho, \quad \widetilde{G}_{\mathrm{B}} = \left(R_2^2 G_{\mathrm{B}}\right)/\kappa. \tag{8}$$

Note that for the governing equations in (4) and (6) we have $u_w(\xi) = u(\rho_{\min})|_\xi$.

Subsequently, we use a general form of (1), (2), (4) and (6) given as follows

$$\begin{cases} \dfrac{du_{\mathrm{B}}}{dx}(x) = \alpha_{1\mathrm{B}} \left(u_w(x) - u_{\mathrm{B}}(x)\right) + \alpha_{2\mathrm{B}}, \quad 0 < x < x_{\max}, \\ u_{\mathrm{B}}(0) = u_{\mathrm{B}0}, \end{cases} \tag{9}$$

$$\begin{cases} \dfrac{d^2u}{dr^2}(r) + \dfrac{1}{r}\dfrac{du}{dr}(r) + \alpha_1 \left(u_a - u(r)\right) + \alpha_2 = 0, \quad b < r < c, \\ \dfrac{du}{dr}(b) = A\left(u(b)|_x - u_{\mathrm{B}}(x)\right), \quad u(c) = u_t, \end{cases} \tag{10}$$

where $u_w(x) = u(b)|_x$. Note that values of the parameters $\alpha_{1\mathrm{B}}$, $\alpha_{2\mathrm{B}}$, $x_{\max}$, $u_{\mathrm{B}0}$, as well as α_1, α_2, u_a, b, c, A, u_t can be calculated on the basis of the formulas describing a given problem in the dimensional or non-dimensional form.

3 Interval Finite Difference Scheme for Solving the Bioheat Transfer Problem

Now we present some conventional finite difference method for solving the bio-heat problem considered. It is partly based on the scheme proposed in Majchrzak (2011). Finally, its interval counterpart is introduced.

3.1 Conventional Finite Difference Method

The approach based on finite differences requires generation of mesh points. Hence, we choose positive integers m, n. Then, we find the stepsizes $k = x_{\max}/m$, $h = (c - b)/n$ and the grid points such that

$$x_j = jk, \quad j = 0, 1, \ldots, m, \qquad r_i = b + ih, \quad i = 0, 1, \ldots, n.$$

Now we express the terms of the governing equation in (10) at the grid points r_i

$$\frac{d^2u(r_i)}{dr^2} + \frac{1}{r_i}\frac{du(r_i)}{dr} + \alpha_1 \left(u_a - u(r_i)\right) + \alpha_2 = 0, \tag{11}$$

where u_a depends on the vessel model used. We take $u_a = u_{\mathrm{B}0}$ for a traversing vessel and $u_a = u_{\mathrm{B}}(x_j)$ for a supplying vessel, respectively.

Then, we use the central finite difference formulas of the second order, together with the appropriate local truncation errors, for $d^2u/dr^2(r_i)$ and du/dr (r_i) in (11). Hence, for $j = 0, 1, \ldots, m$, we have

$$\frac{d^2u\left(r_i\right)}{dr^2} = \frac{u\left(r_{i-1}\right) - 2u\left(r_i\right) + u\left(r_{i+1}\right)}{h^2} - \frac{h^2}{12}\frac{d^4u\left(\xi_{ji}^{(1)}\right)}{dr^4}, \tag{12}$$

$$\frac{du\left(r_i\right)}{dr} = \frac{u\left(r_{i+1}\right) - u\left(r_{i-1}\right)}{2h} - \frac{h^2}{6}\frac{d^3u\left(\xi_{ji}^{(2)}\right)}{dr^3}, \tag{13}$$

where $\xi_{ji}^{(1)}, \xi_{ji}^{(2)} \in \left(r_{i-1}, r_{i+1}\right)|_{x_j}$ and $u = u(r_i)|_{x_j}$. If we insert (12), (13) to (11), then we obtain

$$\begin{aligned}&\overline{\lambda}_i u\left(r_{i-1}\right) + \beta u\left(r_i\right) + \widetilde{\lambda}_i u\left(r_{i+1}\right) = -\gamma_1 u_a + \gamma_2 + \widehat{R}_{ji},\\ &i = 1, 2, \ldots, n-1,\end{aligned} \tag{14}$$

where $\overline{\lambda}_i = 1 - h/\left(2r_i\right)$, $\widetilde{\lambda}_i = 1 + h/\left(2r_i\right)$, $\beta = -2 - \alpha_1 h^2$, $\gamma_1 = \alpha_1 h^2$, $\gamma_2 = -\alpha_2 h^2$,

$$\widehat{R}_{ji} = \frac{h^4}{12}\frac{d^4u\left(\xi_{ji}^{(1)}\right)}{dr^4} + \frac{1}{r_i}\frac{h^4}{6}\frac{d^3u\left(\xi_{ji}^{(2)}\right)}{dr^3}. \tag{15}$$

For the boundary conditions in (10), we have

$$\frac{du\left(r_0\right)}{dr} = A\left(u\left(r_0\right)|_{x_j} - u_{\mathrm{B}}\left(x_j\right)\right), \quad u\left(r_n\right) = u_t. \tag{16}$$

We use the forward finite difference formula of the second order, together with the appropriate local truncation error, for $du/dr(r_0)$ in $(16)_1$. We get

$$(3 + 2Ah)\, u\left(r_0\right) - 4u\left(r_1\right) + u(r_2) = 2Ahu_{\mathrm{B}}(x_j) + \frac{2h^3}{3}\frac{d^3u\left(\xi_j^{(3)}\right)}{dr^3}, \quad u\left(r_n\right) = u_t, \tag{17}$$

where $\xi_j^{(3)} \in \left(r_0, r_1\right)|_{x_j}$, $j = 0, 1, ..., m$.

Secondly, we consider the initial value problem (9). We express the terms of the governing equation at the grid points x_j. Hence, we get

$$\frac{du_{\mathrm{B}}}{dx}\left(x_j\right) = \alpha_{1\mathrm{B}}\left(u_w\left(x_j\right) - u_{\mathrm{B}}\left(x_j\right)\right) + \alpha_{2\mathrm{B}}. \tag{18}$$

First, we use the forward finite difference formula of the first order, together with the appropriate local truncation error, for $du_{\mathrm{B}}/dx\left(x_0\right)$ in (18). We obtain

$$u_{\mathrm{B}}\left(x_1\right) + \left(\alpha_{1\mathrm{B}}k - 1\right)u_{\mathrm{B}}\left(x_0\right) - \alpha_{1\mathrm{B}}ku_w\left(x_0\right) = \alpha_{2\mathrm{B}}k + \frac{k^2}{2}\frac{d^2u_{\mathrm{B}}\left(\eta_0\right)}{dx^2}, \tag{19}$$

where $\eta_0 \in (x_0, x_1)$ and $u_w(x_0) = u(b)|_{x_0}$. Then, we use the central finite difference formula of the second order, together with the appropriate local truncation error, for $du_{\mathrm{B}}/dx(x_j)$, $j = 1, 2, \ldots, m-1$, in (18). We have

$$\begin{aligned} u_{\mathrm{B}}(x_{j+1}) + 2\alpha_{1\mathrm{B}} k u_{\mathrm{B}}(x_j) - u_{\mathrm{B}}(x_{j-1}) &= 2k(\alpha_{1\mathrm{B}} u_w(x_j) + \alpha_{2\mathrm{B}}) \\ &+ \frac{k^3}{3}\frac{d^3 u_{\mathrm{B}}(\eta_j)}{dx^3}, \quad j = 1, 2, \ldots, m-1, \end{aligned} \tag{20}$$

where $\eta_j \in (x_{j-1}, x_{j+1})$, $u_w(x_j) = u(b)|_{x_j}$. Finally, from the initial condition in (9) we have $u_{\mathrm{B}}(x_0) = u_{\mathrm{B}0}$ for $j = 0$.

Consider the formulas (14), (15), (17) and (19), (20). The unknown values of the temperature are $u(r_i)|_{x_j}$, $i = 0, 1, \ldots, n$, $j = 0, 1, \ldots, m$ and $u_{\mathrm{B}}(x_j)$, $j = 1, 2, \ldots, m$. They can be easily calculated, if we solve the system of $(m+1)(n+2)$ linear equations with the same number of unknowns (where the initial condition $u_{\mathrm{B}}(x_0) = u_{\mathrm{B}0}$ is also included in the system), given by the matrix representation of the form

$$Cu = \widehat{E}_C + \widehat{E}_L, \tag{21}$$

where

$$C = \begin{bmatrix} D & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 \\ G & D & 0 & 0 & \ldots & 0 & 0 & 0 & 0 \\ V & H & D & 0 & \ldots & 0 & 0 & 0 & 0 \\ 0 & V & H & D & \ldots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & D & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ldots & H & D & 0 & 0 \\ 0 & 0 & 0 & 0 & \ldots & V & H & D & 0 \\ 0 & 0 & 0 & 0 & \ldots & 0 & V & H & D \end{bmatrix}, \quad u = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{m-3} \\ u_{m-2} \\ u_{m-1} \\ u_m \end{bmatrix}, \tag{22}$$

$$u_j = \left[u_{\mathrm{B}}(x_j),\, u(r_0)|_{x_j},\, u(r_1)|_{x_j},\, \ldots,\, u(r_n)|_{x_j}\right]^{\mathrm{T}}, \quad j = 0, 1, \ldots, m, \tag{23}$$

$$\widehat{E}_C = \left[e_{C0}, e_{C1}, e_{C2}, \ldots, e_{Cm}\right]^{\mathrm{T}}, \quad \widehat{E}_L = \left[e_{L0}, e_{L1}, e_{L2}, \ldots, e_{Lm}\right]^{\mathrm{T}}, \tag{24}$$

and

$$\dim D = \dim G = \dim H = \dim V = (n+2) \times (n+2).$$

The matrices of coefficients G, H and V are all the sparse matrices. The only nonzero values of elements are $g_{00} = \alpha_{1\mathrm{B}} k - 1$, $g_{01} = -\alpha_{1\mathrm{B}} k$, $h_{00} = 2\alpha_{1\mathrm{B}} k$, $h_{01} = -2\alpha_{1\mathrm{B}} k$ and $v_{00} = -1$.

Now let us consider the vectors e_{Lj}, $j = 0, 1, \ldots, m$. Their components include the local truncation error terms of the conventional finite difference schemes at each mesh point. They depend on the stepsizes h and k, values of some derivatives of u and u_{B} at the midpoints and also the distances $r_i|_{x_j}$ from the vessel axis. We have

$$\begin{aligned}
e_{L0} &= \left[0, \frac{2}{3}h^3 \frac{d^3u\left(\xi_0^{(3)}\right)}{dr^3}, \widehat{R}_{01}, \widehat{R}_{02}, \ldots, \widehat{R}_{0\,n-1}, 0\right]^{\mathrm{T}}, \\
e_{L1} &= \left[\frac{k^2}{2}\frac{d^2u_{\mathrm{B}}\left(\eta_0\right)}{dx^2}, \frac{2}{3}h^3 \frac{d^3u\left(\xi_1^{(3)}\right)}{dr^3}, \widehat{R}_{11}, \widehat{R}_{12}, \ldots, \widehat{R}_{1\,n-1}, 0\right]^{\mathrm{T}}, \qquad (25) \\
e_{Lj} &= \left[\frac{k^3}{3}\frac{d^3u_{\mathrm{B}}\left(\eta_{j-1}\right)}{dx^3}, \frac{2}{3}h^3 \frac{d^3u\left(\xi_j^{(3)}\right)}{dr^3}, \widehat{R}_{j1}, \widehat{R}_{j2}, \ldots, \widehat{R}_{j\,n-1}, 0\right]^{\mathrm{T}}, \\
j &= 2, 3, \ldots, m.
\end{aligned}$$

Finally, we focus on the coefficients of the matrix D and the vectors e_{Cj}, $j = 0, 1, \ldots, m$. As we can see in (14), their values depend on the vessel model considered. In the case of a traversing vessel we have

$$D = \begin{bmatrix}
1 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 \\
-2Ah & 3+2Ah & -4 & 1 & \ldots & 0 & 0 & 0 & 0 \\
0 & \overline{\lambda}_1 & \beta & \widetilde{\lambda}_1 & \ldots & 0 & 0 & 0 & 0 \\
0 & 0 & \overline{\lambda}_2 & \beta & \ldots & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \ldots & \beta & \widetilde{\lambda}_{n-3} & 0 & 0 \\
0 & 0 & 0 & 0 & \ldots & \overline{\lambda}_{n-2} & \beta & \widetilde{\lambda}_{n-2} & 0 \\
0 & 0 & 0 & 0 & \ldots & 0 & \overline{\lambda}_{n-1} & \beta & \widetilde{\lambda}_{n-1} \\
0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 1
\end{bmatrix}, \qquad (26)$$

$$\begin{aligned}
e_{C0} = \left[u_{\mathrm{B}0}, 0, \gamma, \gamma, \ldots, \gamma, u_t\right]^{\mathrm{T}}, \quad e_{C1} = \left[\alpha_{2\mathrm{B}}k, 0, \gamma, \gamma, \ldots, \gamma, u_t\right]^{\mathrm{T}}, \\
e_{Cj} = \left[2\alpha_{2\mathrm{B}}k, 0, \gamma, \gamma, \ldots, \gamma, u_t\right]^{\mathrm{T}}, \quad j = 2, 3, \ldots, m,
\end{aligned} \qquad (27)$$

where $\gamma = -(\alpha_1 u_{\mathrm{B}0} + \alpha_2)h^2$. On the other hand, in the case of a supplying vessel, we obtain

$$D = \begin{bmatrix}
1 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 \\
-2Ah & 3+2Ah & -4 & 1 & \ldots & 0 & 0 & 0 & 0 \\
\gamma_1 & \overline{\lambda}_1 & \beta & \widetilde{\lambda}_1 & \ldots & 0 & 0 & 0 & 0 \\
0 & \gamma_1 & \overline{\lambda}_2 & \beta & \ldots & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \ldots & \beta & \widetilde{\lambda}_{n-3} & 0 & 0 \\
0 & 0 & 0 & 0 & \ldots & \overline{\lambda}_{n-2} & \beta & \widetilde{\lambda}_{n-2} & 0 \\
0 & 0 & 0 & 0 & \ldots & \gamma_1 & \overline{\lambda}_{n-1} & \beta & \widetilde{\lambda}_{n-1} \\
0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 1
\end{bmatrix}, \qquad (28)$$

$$e_{C0} = \left[u_{\mathrm{B}0}, 0, \gamma_2, \gamma_2, \ldots, \gamma_2, u_t\right]^{\mathrm{T}}, \quad e_{C1} = \left[\alpha_{2\mathrm{B}}k, 0, \gamma_2, \gamma_2, \ldots, \gamma_2, u_t\right]^{\mathrm{T}}, \\ e_{Cj} = \left[2\alpha_{2\mathrm{B}}k, 0, \gamma_2, \gamma_2, \ldots, \gamma_2, u_t\right]^{\mathrm{T}}, \quad j = 2, 3, \ldots, m. \tag{29}$$

Remark 1. Let u_{ji} and $u_{\mathrm{B}j}$ approximate $u(r_i)|_{x_j}$ and $u_{\mathrm{B}}(x_j)$, respectively. If we also omit the local truncation error terms given in (14), (15), (17) and (19), (20) (which entails neglecting all the components of the vectors e_{Lj}, $j = 0, 1, \ldots, m$), then we get the conventional finite difference method for solving the bioheat transfer problem considered.

3.2 Interval Finite Difference Method

Now we propose the interval finite difference method for solving a given bioheat transfer problem. It is based on the finite difference schemes together with the appropriate local truncation error terms.

The main assumption that has to be made concerns values of the derivatives (at some unknown midpoints) that appear in the terms of the local truncation error. Hence, for the interval approach we assume that there exist the intervals $S_{ji}^{(1)}$, $S_{ji}^{(2)}$, $S_{j}^{(3)}$, $Q^{(1)}$ and $Q_j^{(2)}$, such that the following relations hold

$$\frac{d^4u}{dr^4}\left(\xi_{ji}^{(1)}\right) \in S_{ji}^{(1)} = \left[\underline{S}_{ji}^{(1)}, \overline{S}_{ji}^{(1)}\right], \quad \frac{d^3u}{dr^3}\left(\xi_{ji}^{(2)}\right) \in S_{ji}^{(2)} = \left[\underline{S}_{ji}^{(2)}, \overline{S}_{ji}^{(2)}\right], \\ \xi_{ji}^{(1)}, \xi_{ji}^{(2)} \in \left(r_{i-1}, r_{i+1}\right)|_{x_j}, \quad i = 1, 2, \ldots, n-1, \quad j = 0, 1, \ldots, m, \tag{30}$$

$$\frac{d^3u}{dr^3}\left(\xi_{j}^{(3)}\right) \in S_{j}^{(3)} = \left[\underline{S}_{j}^{(3)}, \overline{S}_{j}^{(3)}\right], \\ \xi_{j}^{(3)} \in \left(r_0, r_1\right)|_{x_j}, \quad j = 0, 1, \ldots, m, \tag{31}$$

$$\frac{d^2u_{\mathrm{B}}}{dx^2}\left(\eta_0\right) \in Q^{(1)} = \left[\underline{Q}^{(1)}, \overline{Q}^{(1)}\right], \quad \eta_0 \in \left(x_0, x_1\right), \tag{32}$$

$$\frac{d^3u_{\mathrm{B}}}{dx^3}\left(\eta_j\right) \in Q_j^{(2)} = \left[\underline{Q}_j^{(2)}, \overline{Q}_j^{(2)}\right], \\ \eta_j \in \left(x_{j-1}, x_{j+1}\right), \quad j = 1, 2, \ldots, m-1. \tag{33}$$

Taking into account the relations (30)–(33) in the formulas (14), (15), (17) and (19), (20), respectively, we obtain the appropriate interval finite difference schemes. The system of (m+1)(n+2) interval linear equations obtained, can be given in the similar matrix form as for the conventional approach. We have

$$CU = E_C + E_L, \tag{34}$$

where

$$U = \left[U_0, U_1, U_2, \ldots, U_m\right]^{\mathrm{T}}, \quad U_j = \left[U_{\mathrm{B}j}, U_{j0}, U_{j1}, \ldots, U_{jn}\right]^{\mathrm{T}}, j = 0, 1, \ldots, m, \tag{35}$$

$$E_C = [E_{C0}, E_{C1}, E_{C2}, \dots, E_{Cm}]^{\mathrm{T}}, \quad E_L = [E_{L0}, E_{L1}, E_{L2}, \dots, E_{Lm}]^{\mathrm{T}}, \tag{36}$$

$$\begin{aligned} E_{L0} &= \left[0, \frac{2}{3}h^3 S_0^{(3)}, \widetilde{R}_{01}, \widetilde{R}_{02}, \dots, \widetilde{R}_{0\,n-1}, 0\right]^{\mathrm{T}}, \\ E_{L1} &= \left[\frac{k^2}{2}Q^{(1)}, \frac{2}{3}h^3 S_1^{(3)}, \widetilde{R}_{11}, \widetilde{R}_{12}, \dots, \widetilde{R}_{1\,n-1}, 0\right]^{\mathrm{T}}, \\ E_{Lj} &= \left[\frac{k^3}{3}Q_{j-1}^{(2)}, \frac{2}{3}h^3 S_j^{(3)}, \widetilde{R}_{j1}, \widetilde{R}_{j2}, \dots, \widetilde{R}_{j\,n-1}, 0\right]^{\mathrm{T}}, \quad j = 2, 3, \dots, m, \end{aligned} \tag{37}$$

and

$$\widetilde{R}_{ji} = \frac{h^4}{12}S_{ji}^{(1)} + \frac{1}{R_i}\frac{h^4}{6}S_{ji}^{(2)}. \tag{38}$$

The components of the vectors E_{Cj}, $j = 0, 1, \dots, m$, depend on the vessel model considered. In the case of a traversing vessel we have

$$\begin{gathered} E_{C0} = [U_{\mathrm{B}0}, 0, \gamma, \gamma, \dots, \gamma, U_t]^{\mathrm{T}}, \quad E_{C1} = [\alpha_{2\mathrm{B}}k, 0, \gamma, \gamma, \dots, \gamma, U_t]^{\mathrm{T}}, \\ E_{Cj} = [2\alpha_{2\mathrm{B}}k, 0, \gamma, \gamma, \dots, \gamma, U_t]^{\mathrm{T}}, \quad j = 2, 3, \dots, m, \end{gathered} \tag{39}$$

and in the case of a supplying vessel, we obtain

$$\begin{gathered} E_{C0} = [U_{\mathrm{B}0}, 0, \gamma_2, \gamma_2, \dots, \gamma_2, U_t]^{\mathrm{T}}, \quad E_{C1} = [\alpha_{2\mathrm{B}}k, 0, \gamma_2, \gamma_2, \dots, \gamma_2, U_t]^{\mathrm{T}}, \\ E_{Cj} = [2\alpha_{2\mathrm{B}}k, 0, \gamma_2, \gamma_2, \dots, \gamma_2, U_t]^{\mathrm{T}}, \quad j = 2, 3, \dots, m. \end{gathered} \tag{40}$$

Note that X_j, $j = 0, 1, \dots, m$, R_i, $i = 0, 1, \dots, n$ and U_t are intervals such that $x_j \in X_j$, $r_i \in R_i$, $u_t \in U_t$. Furthermore, we assume that for the temperature of the blood at entrance $u_{\mathrm{B}}(0)$ and the temperature of the wesel wall $u_w(x_j)$, we have $u_{\mathrm{B}}(0) = u_{\mathrm{B}0} \in U_{\mathrm{B}0}$ and $u_w(x_j) = u(b)|_{x_j} \in U_{wj} = U_{j0}$, respectively.

Remark 2. We assume that the local truncation error of the conventional finite difference schemes concerning both the tissue and the blood vessel, can be bounded by the appropriate intervals (i.e. the relations (30)–(33) hold). If $u_{\mathrm{B}}(x_0) \in U_{\mathrm{B}0}$, $u_t \in U_t$, $x_j \in X_j$, $j = 0, 1, \dots, m$, $r_i \in R_i$, $i = 0, 1, \dots, n$ and the system of interval linear equations (34) can be solved with some direct method, then we can prove that for the interval solutions obtained, we have $u_{\mathrm{B}}(x_j) \in U_{\mathrm{B}j}$, $j = 1, 2, \dots, m$ and $u(r_i)|_{x_j} \in U_{ji}$, $j = 0, 1, \dots, m$, $i = 0, 1, \dots, n$.

3.3 Approximation of the Error Term

Note that it is still an open problem how to compute the endpoints of the intervals $S_{ji}^{(1)}$, $S_{ji}^{(2)}$, $S_j^{(3)}$, $Q^{(1)}$ and $Q_j^{(2)}$. In general, we deal with the similar question in the case of any initial-boundary value problem based on finite differences. In this Section, we propose the same method of the endpoints approximation as in e.g. (Jankowska and Sypniewska-Kaminska 2012; 2013). Such an approach does not guarantee that the exact solution belongs to the interval solution computed. Nevertheless, the numerical experiments, performed in e.g. (Jankowska and Sypniewska-Kaminska 2012; 2013) for the problems with known analytical solutions, confirmed that the interval solutions obtained include the exact ones.

First, we consider the relations (30) with $\xi_{ji}^{(1)}, \xi_{ji}^{(2)} \in (r_{i-1}, r_{i+1})|_{x_j}$. We can choose the endpoints $\underline{S}_{ji}^{(1)}$, $\overline{S}_{ji}^{(1)}$ and $\underline{S}_{ji}^{(2)}$, $\overline{S}_{ji}^{(2)}$ as

$$\underline{S}_{ji}^{(1)} \approx \min\left(S_{j\,i-1}^{(1)*}, S_{ji}^{(1)*}, S_{j\,i+1}^{(1)*}\right), \quad \overline{S}_{ji}^{(1)} \approx \max\left(S_{j\,i-1}^{(1)*}, S_{ji}^{(1)*}, S_{j\,i+1}^{(1)*}\right), \tag{41}$$

$$\underline{S}_{ji}^{(2)} \approx \min\left(S_{j\,i-1}^{(2)*}, S_{ji}^{(2)*}, S_{j\,i+1}^{(2)*}\right), \quad \overline{S}_{ji}^{(2)} \approx \max\left(S_{j\,i-1}^{(2)*}, S_{ji}^{(2)*}, S_{j\,i+1}^{(2)*}\right), \tag{42}$$

where

$$S_{ji}^{(1)*} = \frac{d^4u}{dr^4}(r_i)\,|_{x_j}, \quad S_{ji}^{(2)*} = \frac{d^3u}{dr^3}(r_i)\,|_{x_j}. \tag{43}$$

For the relation (31) with $\xi_j^{(3)} \in (r_0, r_1)|_{x_j}$, we choose the endpoints $\underline{S}_j^{(3)}$, $\overline{S}_j^{(3)}$ as

$$\underline{S}_j^{(3)} \approx \min\left(S_{j\,0}^{(3)*}, S_{j\,1}^{(3)*}\right), \quad \overline{S}_j^{(3)} \approx \max\left(S_{j\,0}^{(3)*}, S_{j\,1}^{(3)*}\right) \tag{44}$$

where

$$S_{ji}^{(3)*} = \frac{d^2u}{dr^2}(r_i)\,|_{x_j}, \quad i = 0, 1. \tag{45}$$

Finally, we consider the relation (32) with $\eta_0 \in (x_0, x_1)$ and the relation (33) with $\eta_j \in (x_{j-1}, x_{j+1})$. We choose the endpoints $\underline{Q}^{(1)}$, $\overline{Q}^{(1)}$ and $\underline{Q}_j^{(2)}$, $\overline{Q}_j^{(2)}$, as

$$\underline{Q}^{(1)} \approx \min\left(Q_0^{(1)*}, Q_1^{(1)*}\right), \quad \overline{Q}^{(1)} \approx \max\left(Q_0^{(1)*}, Q_1^{(1)*}\right), \tag{46}$$

$$\underline{Q}_j^{(2)} \approx \min\left(Q_{j-1}^{(2)*}, Q_j^{(2)*}, Q_{j+1}^{(2)*}\right), \quad \overline{Q}_j^{(2)} \approx \max\left(Q_{j-1}^{(2)*}, Q_j^{(2)*}, Q_{j+1}^{(2)*}\right), \tag{47}$$

where

$$Q_j^{(1)*} = \frac{d^2u_{\mathrm{B}}}{dx^2}(x_j), \quad j = 0, 1, \qquad Q_j^{(2)*} = \frac{d^3u_{\mathrm{B}}}{dx^3}(x_j). \tag{48}$$

Remark 3. Before we use the interval method for solving the bioheat transfer problem considered with the approximation of the endpoints of the error term intervals, the preliminary stage of computing approximate values of the temperature with the conventional methods described in Sect. 3.1 is required. We obtain u_{ji} that approximate $u(r_i)|_{x_j}$ for the tissue and $u_{\mathrm{B}j}$ that approximate $u_{\mathrm{B}}(x_j)$ for the blood, where $u_{\mathrm{B}0}$ is known. After that we can approximate values of $d^4u(r_i)/dr^4|_{x_j}$ and $d^3u(r_i)/dr^3|_{x_j}$ given in $(43)_1$ and $(43)_2$ with the appropriate finite differences (FD) of the second order in the following way: the forward FD formula for $i = 1, 2$; the central FD formula for $i = 3, 4, \ldots, n-3$; the backward FD formula for $i = n-2, n-1$. Then, we can apply the forward FD formula of the second order for the approximation of $d^2u(r_i)/dr^2|_{x_j}$ given in (45). In the case of the derivative $d^2u_{\mathrm{B}}(x_j)/dx^2$ in $(48)_1$ we can use the forward FD formula of the second order. Finally, we approximate $d^3u_{\mathrm{B}}(x_j)/dx^3$ given in $(48)_3$ with the FD formulas of the second order as follows: the forward FD formula for $j = 1, 2$; the central FD for $j = 3, 4, \ldots, m-3$; the backward FD formula for $j = m-2, m-1$.

4 Numerical Experiment

For the present numerical experiment we choose values of the parameters as in e.g. (Huang et al. 1994). We have $\lambda = \lambda_{\mathrm{B}} = 0.5\,[\mathrm{W/(m{\cdot}K)}]$, $c = c_{\mathrm{B}} = 3900\,[\mathrm{J/(kg{\cdot}K)}]$, $\rho = \rho_B = 1060\,[\mathrm{kg/m^3}]$, $\dot{Q}_{met} = 1000\,[\mathrm{W/m^3}]$, $\dot{Q}_{\mathrm{B}met} = 500\,[\mathrm{W/m^3}]$, $W_{\mathrm{B}} = 10\,[\mathrm{kg/(m^3{\cdot}s)}]$, $v_{\mathrm{B}} = 3.023\mathrm{E}\text{-}3\ [\mathrm{m/s}]$, $\alpha = 500\,[\mathrm{W/(m^2{\cdot}K)}]$. We also take $w_{\mathrm{B}0} = w_t = w_{\mathrm{ref}} = 37\ [^\circ\mathrm{C}]$ and $R_1 = 0.002\,[\mathrm{m}]$, $R_2 = 0.02\,[\mathrm{m}]$, $L = 0.1\,[\mathrm{m}]$. Furthermore, the traversing vessel model is chosen. Subsequently, we use the interval finite difference method given in the matrix representations (34). Note that computations are performed for the problem formulated in the dimensionless coordinates. Then, the results are calculated back to the dimensional coordinates and presented in the paper. In Figs. 2, 3, we can see the temperature distribution and the widths of the interval results in the case of the tissue, respectively.

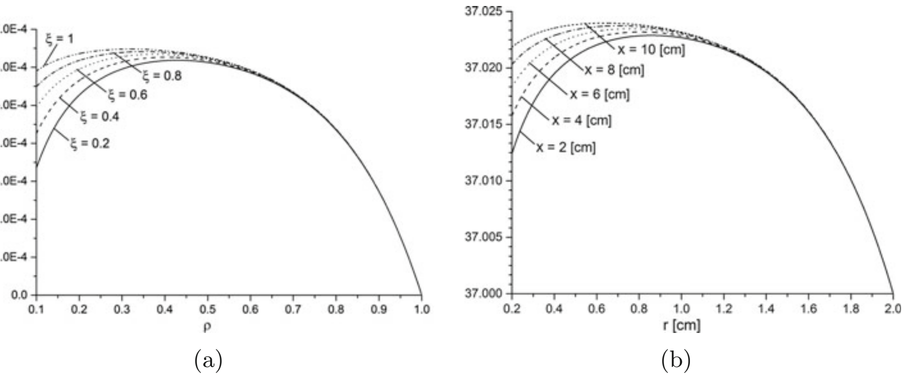


Fig. 2. Temperature distribution: (a) $u = u(\rho)$; (b) $w = w(r)$ in the tissue for the selected positions along the blood vessel axis.

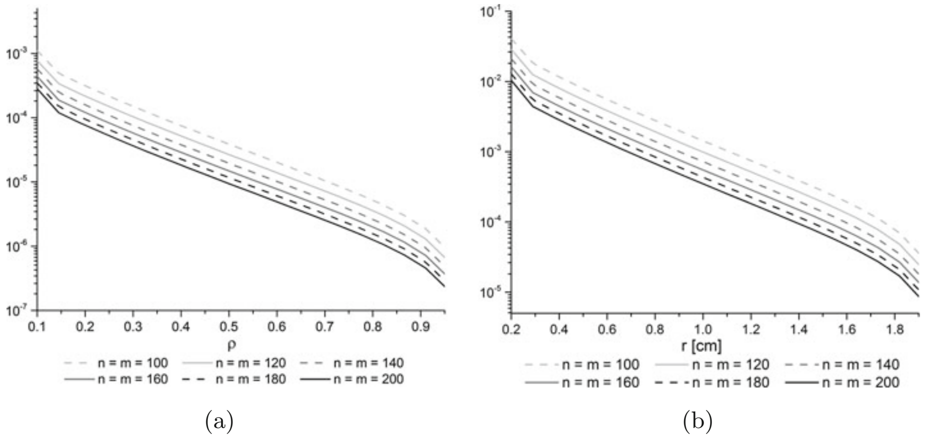


Fig. 3. Widths of the interval solutions: (a) $u = u(\rho)|_{\xi=1}$; (b) $w = w(r)|_{x=10[cm]}$ obtained with the interval method for different values of the constants n and m.

5 Conclusions

The objective of the paper is to propose the interval method for solving the problem of bioheat transfer. An interval approach seems to be of special interest in the area of biomechanics considered. It is due to the fact that such problems often take into account values of biomaterial parameters which are dependent on some environmental factors that influence an individual human or an animal, e.g. age, state of health, lifestyle. Hence, we usually know a range of values instead of some precise value of a given parameter. The main advantage of an interval approach is the ability to represent the uncertain values in form of intervals. Then, interval solutions obtained with the interval methods include all values that can be taken by physical parameters occurring in the problem formulation.

References

Chato, J.: Heat transfer to blood vessels. J. Biomech. Eng. **5**(102), 110–118 (1980)

Huang, H.W., Chan, C.L., Roemer, R.B.: Analytical solutions of Pennes bio-heat transfer equation with a blood vessel. J. Biomech. Eng. **116**, 208–212 (1994)

Jankowska, M.A.: Remarks on algorithms implemented in some C++ libraries for floating-point conversions and interval arithmetic. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009, Part II. LNCS, vol. 6068, pp. 436–445. Springer, Heidelberg (2010)

Jankowska, M.A., Sypniewska-Kaminska, G.: An interval finite difference method for the bioheat transfer problem described by the Pennes equation with uncertain parameters. Mech. Control **31**(2), 77–84 (2012)

Jankowska, M.A., Sypniewska-Kamiska, G.: Interval finite-difference method for solving the one-dimensional heat conduction problem with heat sources. In: Manninen, P., Öster, P. (eds.) PARA 2012. LNCS, vol. 7782, pp. 473–488. Springer, Heidelberg (2013)

Majchrzak, E., Mochnacki, B.: Numerical model of heat transfer between blood vessel and biological tissue. Comput. Assist. Mech. Eng. Sci. **6**, 439–447 (1999)

Majchrzak, E., Mochnacki, B.: Analysis of Bio-Heat Transfer in the System of Blood Vessel-Biological Tissue, pp. 201–211. Kluwer, Boston (2001)

Majchrzak, E.: Modelling and analysis of thermal phenomena. In: Bedzinski, R. (ed.) "Biomechanics", Section 4 [in polish], pp. 223–361. Institute of Fundamental Technological Research, Polish Academy of Sciences, Warsaw (2011)

Marciniak, A., Szyszka, B.: A central-backward difference interval method for solving the wave equation. In: Manninen, P., Öster, P. (eds.) PARA 2012. LNCS, vol. 7782, pp. 518–527. Springer, Heidelberg (2013)

Workshop on Complex Collective Systems

Bridging the Gap: From Cellular Automata to Differential Equation Models for Pedestrian Dynamics

Felix Dietrich[1], Gerta Köster[2], Michael Seitz[2], and Isabella von Sivers[2](✉)

[1] Technical University of Munich, Boltzmannstr. 3, 85747 Garching, Germany
felix.dietrich@tum.de

[2] Munich University of Applied Sciences, Lothstr. 64, 80335 Münich, Germany
{koester,m.seitz,isabella.von_sivers}@hm.edu

Abstract. Cellular automata (CA) and ordinary differential equation (ODE) based models compete for dominance in microscopic pedestrian dynamics. Both are inspired by the idea that pedestrians are subject to forces. However, there are two major differences: In a CA, movement is restricted to a coarse grid and navigation is achieved directly by pointing the movement in the direction of the forces. Force based ODE models operate in continuous space and navigation is computed indirectly through the acceleration vector. We present two models emanating from the CA and ODE approaches that remove these two differences: the Optimal Steps Model and the Gradient Navigation Model. Both models are very robust and produce trajectories similar to each other, bridging the gap between the older models. Both approaches are grid-free and free of oscillations, giving cause to the hypothesis that the two major differences are also the two major weaknesses of the older models.

Keywords: Cellular automata · Ordinary differential equation · Pedestrian dynamics · Optimal step model · Gradient Navigation Model

1 Introduction

Several approaches for modeling pedestrian dynamics have been developed in the last decades [1–4]. Among these, two compete for supremacy: cellular automata (CA) and models based on ordinary differential equations (ODE). In typical formulations, both model types use the idea that pedestrians are driven by repulsive and attractive forces. Other pedestrians and obstacles repel, targets attract. However, the mathematical formulations differ fundamentally, especially as far as treatment of space and navigation is concerned.

In CA, the given area is divided into cells of equal shape and area that are either empty or occupied by a pedestrian, a target or an obstacle. This status is updated at each time step, that is, virtual pedestrians move from cell to cell according to certain rules. Typically, the pedestrians navigate along a floor field that expresses attractive and repulsive forces acting on the pedestrians [5,6]. Acceleration to each pedestrian's free-flow velocity is achieved instantaneously if there is space to move. The coarse discretization of space limits the choice of direction and influences space requirements and the handling of speed [6]. Advantages are high computational speed, simplicity, as well as easy and intuitive integration of rules governing pedestrian behavior [7–10].

ODE-models for pedestrian motion are usually inspired by Newtonian mechanics. They also consider attractive and repulsive forces but operate in continuous space and time. Navigation is realized indirectly by computing an acceleration vector from a superposition of forces. Acceleration is delayed by a friction term. The best known ODE-model is the Social Force Model (SFM) introduced by Dirk Helbing and Péter Molnár in 1995 [11]. Problems of this ansatz include inertia and oscillations [12] as well as numerical pitfalls [13]. Furthermore, typical or specific behavior of pedestrians, even prevention of overlapping, can only be achieved by introducing extra complexity [14,15].

In this paper, we present and compare two new models that remove the major differences between the CA and ODE models: First, the Optimal Steps Model [4, 16], which remains rule based as a CA but allows movement in continuous space. Second, the Gradient Navigation Model [17], which uses ordinary differential equations like the SFM but computes velocity, and hence direction of movement, directly from the forces. See Fig. 1 for a schematic representation of how to bridge the gap between CA and ODE models. Both approaches maintain the advantages of the models they were inspired by, but do not suffer from the main disadvantages. They are robust and successfully validated according to the guidelines given in [18]. Both can be calibrated to a given fundamental diagram [16,17]. Numerical experiments yield very similar trajectories showing that they are indeed more alike than the original CA and ODE models. See Sect. 4.

The paper is structured as follows: in Sects. 2 and 3 we briefly introduce the two new models stating their main ideas and underline where they deviate from the CA and social force approaches. For detailed descriptions we refer to the original publications [4,16,17]. Then we show the results of numerical experiments for all four model types for all four model types to support our hypothesis that the two new models do not only perform better but produce similar results (Sect. 4). In Sect. 5 we discuss which differences remain. In the conclusion section (Sect. 6) we propose desirable next steps.

2 Optimal Steps Model

In this section we give an outline of the Optimal Steps Model (OSM) as it is described in [4] and enhanced in [16]. The model is inspired by the idea that pedestrians try to optimize their position in space according to a balance of goals: reaching the target and avoiding obstacles and other pedestrians. This approach is also used for cellular automata models of pedestrian movement [3]. Virtual pedestrians move by locally minimizing a scalar field $P : \mathbb{R}^2 \to \mathbb{R}$ that maps each position to a value that is computed as a superposition of potentials: attraction

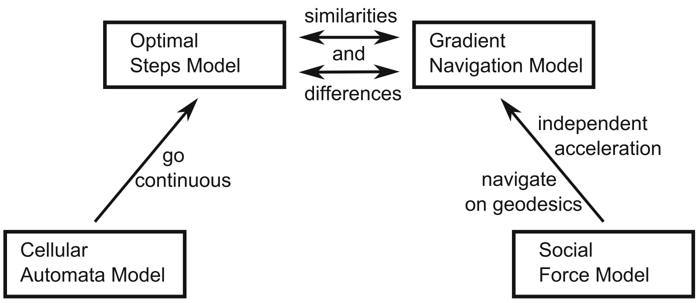


Fig. 1. Bridging the gap between CA- and DE-models

by targets and repulsion by other pedestrians and obstacles. The aggregated potential for each pedestrian serves as an objective function to find the optimal position on a disc. The radius of the disc is the stride length that corresponds to each pedestrian's individual free-flow velocity (Fig. 3). Thus, stepping forward has become a non-linear optimization problem in continuous space. As a result, pedestrians can make adjusting steps that are shorter than the free-flow stride length, thus naturally slowing down in dense situations when navigation becomes difficult (see Fig. 2). Calibration to a given density-velocity profile is achieved by adjusting the repulsive potentials of pedestrians and obstacles [16]. Single steps can still be distinguished in the trajectories (see Fig. 2). That is why we call the model quasi-continuous. In contrast to a CA, the pedestrians are not fixed to cells, but can reach every location in the observed area. Discretization has become a dynamic process governed by natural stepping behavior, which represents a natural discretization of pedestrian movement.

If we restrict the optimization to positions on the circle, instead of on the whole disc, and allow only six equidistant positions on the circle, a hexagonal CA grid is reproduced. With four or eight positions we get rectangular CA grids with von-Neumann or Moore neighborhoods [4].

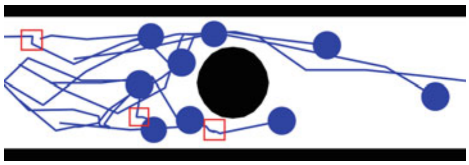


Fig. 2. Pedestrians simulated by the OSM move simultaneously from left to right along a corridor with a column placed in the middle. They make small, sometimes evasive, steps as highlighted by the (red) rectangles. Once the obstacle has been passed, the stride length increases and the pedestrians resume their free-flow velocity (Color figure online).

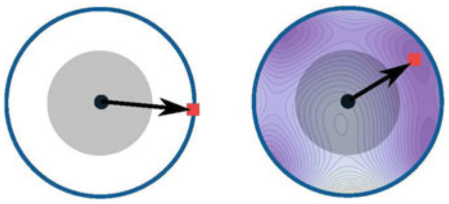


Fig. 3. The two step types in the OSM: the gray region shows a pedestrian torso with its center. The individual stride length is given by the solid (blue) circular line. The square displays the minimum of the aggregated potential for the pedestrian and hence the next position. On the left, it is on the circular line, that is, the pedestrian strides freely. On the right, the next position is within the circle (Color figure online).

2.1 Potential

A very successful way to compute a target potential is to compute the arrival time of a wave front propagating from the target and thereby skirting obstacles. As a result, pedestrians navigate along geodesics [5,19]. Let $\Omega \subset \mathbb{R}^2$ be the area of the scenario and $\Gamma \subset \partial\Omega$ the boundary of the target region. Then the Eikonal equation defines the arrival time $\sigma : \Omega \to \mathbb{R}$ of a wave front propagating with speed $F(x)$:

$$\begin{aligned} F(x)\|\nabla\sigma(x)\| &= 1 \qquad && \text{for } x \in \Omega \\ \sigma(x) &= 0 \qquad && \text{for } x \in \Gamma \end{aligned} \tag{1}$$

We compute its numerical solution σ_N by Sethian's Fast Marching algorithm on a two-dimensional grid [20,21]. Between the distinct values of the grid, σ_N is interpolated bilinearly [4,5] to $\tilde{\sigma_N}$ in the OSM. Thus the target potential $P_t(x) = \tilde{\sigma_N}(x)$ is given for each point $x \in \Omega$.

In addition to the target potential, the aggregated potential $P_i(x)$ at a given point $x \in \Omega$ is composed of pedestrian potentials and obstacle potentials [4]. The pedestrian potential $P_p^j(x)$ is generated by pedestrian j. It only depends on the Euclidean distance between the center of pedestrian j and the considered position x in the scenario. The obstacle potential is very similar to the pedestrian potential. Here, the obstacle potential $P_o^k(x)$ for obstacle k depends on the Euclidean distance between the considered position x and the nearest point of the obstacle to x.

For each pedestrian i in a scenario with n pedestrians and m obstacles

$$P_i(x) = P_t(x) + \sum_{j=1, j\neq i}^{n} P_p^j(x) + \sum_{k=1}^{m} P_o^k(x), \tag{2}$$

assigns the aggregated potential $P_i(x)$ to an arbitrary point $x \in \Omega$ [4].

3 Gradient Navigation Model

The Gradient Navigation Model (GNM) [17] introduces a new set of ordinary differential equations to determine the position of each pedestrian x_i in two dimensional space as well as the scalar speed and navigational direction [17]. The idea is to compute the velocity vector as the gradient of several distance dependent, scalar functions similar to P_i in the OSM (see Eq. 2). This constitutes a strong deviation from Newtonian dynamics and hence from the Social Force Model [11], where the acceleration vector is computed from the forces.

The change in position of pedestrians is not instantaneous. We follow [22] and assume a certain time delay. We model the resulting relaxed speed adaptation by a multiplicative, time dependent, scalar variable w, which we call relaxed speed. Its derivative with respect to time, $\dot{w}$, is similar to acceleration.

With initial conditions $\boldsymbol{x}_0 = \boldsymbol{x}(0)$ and $w_0 = w(0)$ the Gradient Navigation Model is given by

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= w(t)\mathbf{N}(\mathbf{x}, t) \\ \dot{w}(t) &= \tfrac{1}{\tau}\left(v(\rho(\mathbf{x}))\|\mathbf{N}(\mathbf{x}, t)\| - w(t)\right) \end{aligned} \tag{3}$$

with navigation function $\mathbf{N}$

$$\mathbf{N}(\mathbf{x}, t) = -g\left(g(\nabla\sigma(\mathbf{x})) + g(\nabla\delta(\mathbf{x}, t))\right) \tag{4}$$

Function $g : \mathbb{R}^2 \to \mathbb{R}^2$ scales the length of a given vector to lie in the interval $[0, 1]$. For the exact formula see [17]. The position $\boldsymbol{x} : \mathbb{R} \to \mathbb{R}^2$ and the one-dimensional relaxed speed $w : \mathbb{R} \to \mathbb{R}$ are functions of time t.

The individuals' desired speed is represented by $v(\rho(\mathbf{x}))$, which can be chosen to enforce additional deceleration in a dense crowd as observed by [23]. Note that this does not lead to an exact match of the simulated speed-density relation with the fundamental diagram. Even with a constant $v(\rho(\mathbf{x}))$ as chosen in this paper, pedestrians in the GNM slow down when they approach others.

Both σ and δ are closely related to the potential functions of the OSM described in Eq. 2. The first arrival time to the closest target region around static obstacles is again given by $\sigma : \Omega \to \mathbb{R}$ from Eq. 1. The gradients of all distance functions of all other pedestrians and walls are combined and form $\nabla\delta$:

$$\nabla\delta_i(x) = \sum_{j=1, j\neq i}^{n} \nabla P_p^j(x) + \sum_{k=1}^{m} \nabla P_o^k(x) \tag{5}$$

With these equations, the direction of pedestrian i changes independently of physical constraints, similar to the idea of a heuristic by [24] and the Optimal Steps Model [4,16]. The norm of the navigation function N and the relaxed speed w determines the speed $\dot{x}$ in the desired direction.

The norm of the gradient ∇P_p^j resembles the monotonically decreasing function used for the potential values in the OSM. To ensure smoothness of the derivatives, we choose a smooth exponential function with compact support:

$$\|\nabla P_p^j\| = \begin{cases} p^{\max}\exp\left(\frac{1}{(r/R)^2 - 1}\right) & |r/R| < 1 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

with constants $p^{\max} > 0$ and $R > 0$ and $r = \|x_i - x_j\|$ (see Fig. 4). The gradients of the potentials of obstacles use the same formula, only the values of the constants R and $p^{\max}$ differ.

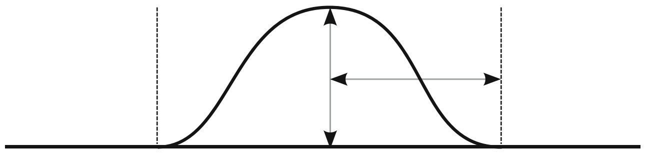


Fig. 4. The norm of ∇P_p^j, which is determined by its height p^{max} and width R.

4 Results

Grid restrictions in CA models and indirect navigation through acceleration in the ODE models constitute the major differences in the model formulations. In two computer experiments described below, we demonstrate how pedestrian trajectories become more similar when these differences are removed with the OSM and the GNM. In the first scenario, a single pedestrian leans against a wall from where he or she moves to a target located to the right (see Fig. 5). The CA and SFM produce unnatural behavior: in the CA model, the hexagonal grid is evident. In the SFM the pedestrian moves in a wide arc and then circles around the target. In fact, depending on the numerical method, the pedestrian may never reach the goal or slow down [13]. The relaxation constant $\tau = 0.5$ (seconds) is chosen similar to values reported in [11,22]. The unnatural circling can be somewhat mitigated, at the cost of extra complexity, by attenuating the speed in a close vicinity of the target. In the OSM, the pedestrian steps away from the wall and at the same time towards the goal. The second and third steps are straight towards the goal, where the pedestrian stops. In the GNM, the pedestrian first moves a little away from the wall and then towards the target in a smooth curve parallel to the wall. There are no oscillations present. The pedestrian comes to a halt at the target. The two trajectories look very similar.

The second scenario is inspired by a laboratory experiment by Moussaïd [22]: Trajectories of pedestrians are observed who move around a stationary person acting as pillar in the middle of a corridor. In our computer experiment we use a pillar, that is, an obstacle and not a virtual person (Fig. 6). Pedestrians walk from left to right in a sequential order, so that they do not impede each other. Their start and target positions are chosen randomly at the ends of the corridor but not too close to the wall. The opening between the wall and a pillar in the center corresponds exactly to the torso diameter of a virtual pedestrian. Once again the hexagonal grid of the CA looks unnatural. On a larger scale this may become unimportant, but it makes fine resolution of bottlenecks impossible.

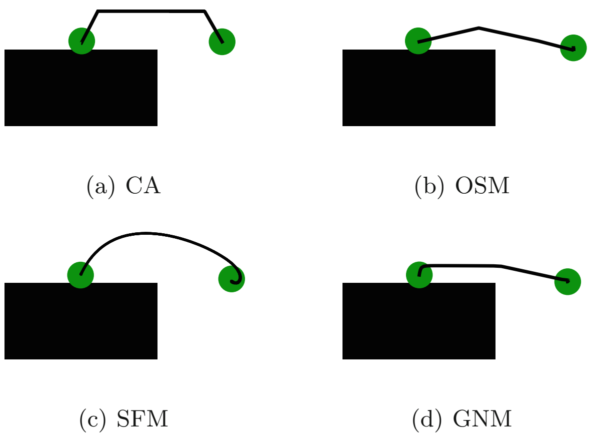


Fig. 5. Trajectories of a single pedestrian initially positioned close to a wall (2×1 m) who moves towards a target 2 m to the right. Stride length for CA: 0.39 m (corresponding to the cell diameter), maximum stride length for OSM: 0.70 m (average for speed 1.34 m/s [4]), relaxation constant τ in SFM / GNM: 0.5 s.

The trajectories in the SFM are smooth. An artifact can be observed shortly after the pedestrians have passed the pillar: They bounce back from the walls. This oscillation is typical for Newtonian systems where the velocity vector is computed by integrating the acceleration vector that depends on the directional force (here along geodesics) and, in our scenario, the forces from the pillar and the wall. The OSM and GNM show similar smooth trajectories that roughly form an eye around the pillar. If a magnifying lens was used, individual steps would be visible in the OSM.

5 Remaining Differences

The OSM with its quasi-continuous dynamic discretization of space has drawn nearer to continuous models, while the GNM has adopted the navigational ideas of successful CA-models and the OSM and integrated them in an ODE context. Nevertheless, some differences remain. In the OSM, there is no acceleration phase. Pedestrians reach their desired free flow velocity instantaneously if they are not hampered by preceding pedestrians or obstacles. In the GNM, acceleration is relaxed. A specific density-velocity relation can be set through direct adjustment of $v(\rho)$ in the relaxed speed equation. The density-velocity dependency in the OSM results from proper numerical calibration of the repulsive pedestrian potential, which reflects an individual's personal need for private space. The trajectories of the OSM capture individual steps (Fig. 5(b)) whereas the GNM focuses on the position of the center of mass (Fig. 5(d)).

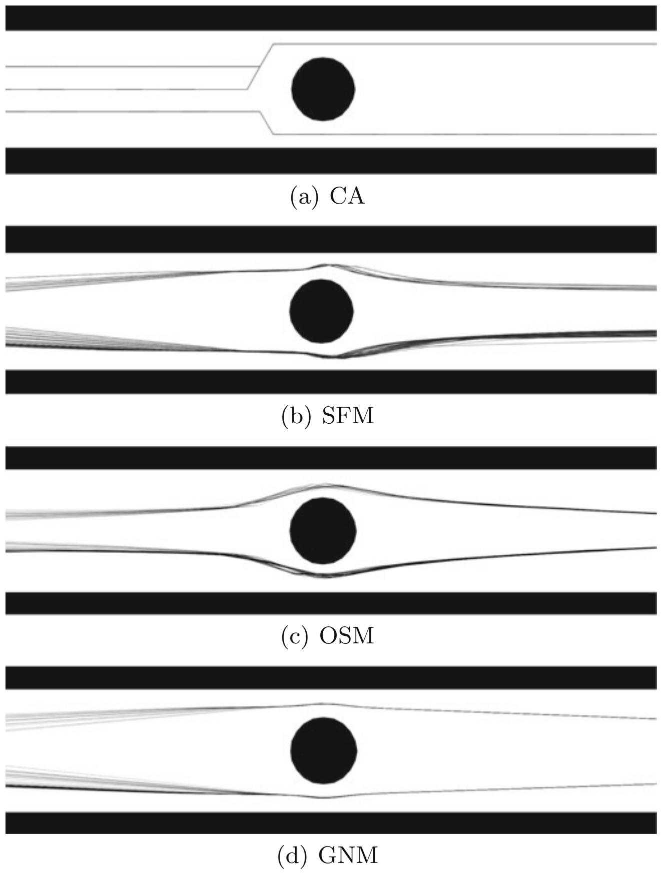


Fig. 6. Trajectories for pedestrians passing from left to right. Pedestrians walk around a pillar; the opening between the wall and the pillar is exactly the torso diameter.

6 Conclusion and Future Work

Two new models for pedestrian motion were presented, one emanating from rule based CA models, but with continuous treatment of space and one based on ODEs using navigation on geodesics as in CA models. The trajectories of the new models are quite similar, thus bridging the gap between the traditional modeling approaches. Closing the gap entirely may well be possible, but entails more stringent mathematical formulations and studies of the OSM to complement the present algorithmic formulation and to allow rigorous proofs.

Comparing simulation models of pedestrian movement is important in order to classify them and finally select the appropriate model for a given task. Applications vary from few pedestrians to huge crowds, from real time requirements on personal computers to massive parallel computations on mainframes. Therefore, choosing the right model with its advantages and disadvantages contributes significantly to practical applications.

We will continue to explore the model similarities and differences trying to pin down the mathematical causes and to clearly identify effects. A long-term goal could be to express the GNM as the limiting case of the OSM when the discretization in time, Δt, goes to zero.

Acknowledgments. This work was funded by the German Federal Ministry of Education and Research through the project MEPKA on mathematical characteristics of pedestrian stream models (17PNT028).

References

1. Zheng, X., Zhong, T., Liu, M.: Modeling crowd evacuation of a building based on seven methodological approaches. Build. Environ. **44**(3), 437–445 (2009)
2. Smith, A., James, Ch., Jones, R., Langston, P., Lester, E., Drury, J.: Modelling contra-flow in crowd dynamics dem simulation. Saf. Sci. **47**(3), 395–404 (2009)
3. Köster, G., Seitz, M., Treml, F., Hartmann, D., Klein, W.: On modelling the influence of group formations in a crowd. Contem. Soc. Sci. **6**(3), 397–414 (2011)
4. Seitz, M.J., Köster, G.: Natural discretization of pedestrian movement in continuous space. Phys. Rev. E **86**, 046108 (2012)
5. Hartmann, D.: Adaptive pedestrian dynamics based on geodesics. New J. Phys. **12**, 043032 (2010)
6. Köster, G., Hartmann, D., Klein, W.: Microscopic pedestrian simulations: from passenger exchange times to regional evacuation. In: Hu, B., Morasch, K., Pickl, S., Siegle, M. (eds.) Operations Research Proceedings 2010: Selected Papers of the Annual International Conference of the German Operations Research Society, pp. 571–576. Springer (2011)
7. Burstedde, C., Klauck, K., Schadschneider, A., Zittartz, J.: Simulation of pedestrian dynamics using a two-dimensional cellular automaton. Phys. A Stat. Mech. App. **295**, 507–525 (2001)
8. Kirchner, A., Klüpfel, H., Nishinari, K., Schadschneider, A., Schreckenberg, M.: Simulation of competitive egress behavior: comparison with aircraft evacuation data. Physica A **324**(3–4), 689–697 (2003)
9. Henein, C.M., White, T.: Macroscopic effects of microscopic forces between agents in crowd models. Physica A **373**, 694–712 (2007)
10. Ezaki, T., Yanagisawa, D., Ohtsuka, K., Nishinari, K.: Simulation of space acquisition process of pedestrians using proxemic floor field model. Physica A **391**(1–2), 291–299 (2012)
11. Helbing, D., Molnár, P.: Social force model for pedestrian dynamics. Phys. Rev. E **51**(5), 4282–4286 (1995)
12. Chraibi, M.: Validated force-based modeling of pedestrian dynamics. Ph.D. thesis, Universität zu Köln (2012)
13. Köster, G., Treml, F., Gödel, M.: Avoiding numerical pitfalls in social force models. Phys. Rev. E **87**(6), 063305 (2013)
14. Chraibi, M., Seyfried, A., Schadschneider, A.: Generalized centrifugal-force model for pedestrian dynamics. Phys. Rev. E **82**(4), 046111 (2010)

15. Chraibi, M., Kemloh, U., Schadschneider, A., Seyfried, A.: Force-based models of pedestrian dynamics. Netw. Heterogen. Media **6**(3), 425–442 (2011)
16. von Sivers, I.: Numerische Methoden zur Optimierung der Schrittrichtung und -weite in einem Modell der Personenstromsimulation. Master's thesis, Fernuniversität in Hagen (2013)
17. Dietrich, F.: An ode-based model for pedestrian motion and navigation. Bachelor's thesis, Technische Universität München (2013)
18. RiMEA. Richtlinie für Mikroskopische Entfluchtungsanalysen - RiMEA. RiMEA e.V., 2.2.1 edn. (2009)
19. Treuille, A., Cooper, S., Popović, Z.: Continuum crowds. ACM Trans. Graph. **25**(3), 1160–1168 (2006). (SIGGRAPH 2006)
20. Sethian, J.A.: A fast marching level set method for monotonically advancing fronts. Proc. Nat. Acad. Sci. **93**(4), 1591–1595 (1996)
21. Sethian, J.A.: Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science. Cambridge University Press, Cambridge (1999)
22. Moussaïd, M., Helbing, D., Garnier, S., Johansson, A., Combe, M., Theraulaz, G.: Experimental study of the behavioural mechanisms underlying self-organization in human crowds. Pap. Proc. R. Soc. B: Biol. Sci. **276**, 2755–2762 (2009)
23. Weidmann, U.: Transporttechnik der Fussgänger, Schriftenreihe des IVT, vol. 90. Institut für Verkehrsplanung, Transporttechnik, Strassen- und Eisenbahnbau (IVT) ETH, Zürich, 2 edn. (1992)
24. Moussaïd, M., Helbing, D., Theraulaz, G.: How simple rules determine pedestrian behavior and crowd disasters. Proc. Nat. Acad. Sci. **108**(17), 6884–6888 (2011)

Cellular Model of Pedestrian Dynamics with Adaptive Time Span

Marek Bukáček, Pavel Hrabák(✉), and Milan Krbálek

Faculty of Nuclear Sciences and Physical Engineering,
Czech Technical University in Prague, Prague, Czech Republic
{bukacma2,pavel.hrabak,milan.krbalek}@fjfi.cvut.cz

Abstract. A cellular model of pedestrian dynamics based on the Floor Field model is presented. Contrary to the parallel update in Floor Field, the concept of adaptive time span is introduced. This concept, together with the concept of bounds, supports the spontaneous line formation and chaotic queue in front of the bottleneck. Model simulations are compared to the experiment "passing through", from which a phase transition from low to high density is observed.

Keywords: Pedestrian dynamics · Experimental study of phase transition · Adaptive time span

1 Introduction

This article presents a concept of adaptive time span for the class of cellular models of pedestrian dynamics. Adaptive time span is supported by the principle of bounds, which enables the agent to choose an unoccupied cell. Such modification is motivated by the spontaneous line formation in front of the exit during non-panic egress situation partially discussed in [1].

Constructed model is used to simulate phase transition from low to high density in an open room with one entrance and one exit inspired by [2]. This study is supported by the experimental data, which were analyzed from the qualitative point of view. Qualitative analysis of experimental data and model simulations is based on [3] and [4].

Introduced model comes out of the idea of the Floor Field model ([5,6]) and its implementation in F.A.S.T ([7,8]). Mainly, the concept of static potential field generated by the exit is adopted. The Floor Field platform has been supplemented by non-trivial updating scheme which will be presented within the scope of next section. To handle the symmetry problem (discussed e.g. in [7,9], or [10]) time and probability penalization of diagonal movement is implemented. Inspired by [11,12], and [13], simple movement prediction is taken into account.

Principles of adaptive time span and bounds presented in this article were introduced to support the microscopic experimental study of pedestrian motion in lines in front of the bottleneck. As it is shown in Fig. 1, the spontaneous

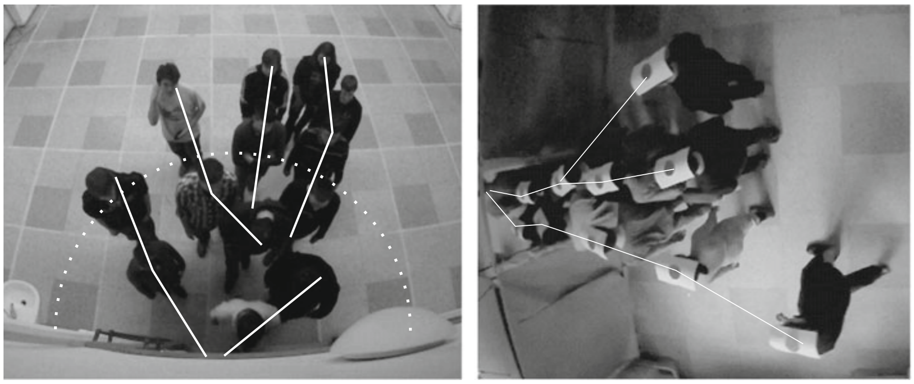


Fig. 1. Spontaneous line formation in front of the exit observed during experiments.

development of lines was observed. Pedestrians preferred the motion in lines to running around the crowd and fighting at the door. The motion in lines is synchronous for short period of time. After certain period without motion, the line disolves and another substituent lines are formed.

2 Definition of Model Dynamics

As usual, the space is divided into square cells by rectangular lattice L with lattice constant equal to 0.5 m. Let us denote the cell x by the position of the cell center $x = (x_1, x_2)$, where the scale unit of x_1 and x_2 axis corresponds to the lattice constant, i.e, x_1 and x_2 are integer numbers, denoting the position of the cell in relation to the origin (0,0) in number of cells. Every cell may be occupied by one agent or empty. The actual state of the lattice L at time t is given by the occupation identifier $\eta(t) = (\eta_x(t))_{x\in L}$, where $\eta_x(t) = 1$ for an occupied cell and $\eta_x(t) = 0$ for an empty cell.

Consider $N \in \mathbb{N}$ agents moving along L by hopping from one cell to another. Let us denote $\xi_i(t) \in L$ the position of the agent $i \in \hat{N}$ at time t, where $\hat{N} = \{1, 2, \ldots, N\}$ is the set of all agents. At first, we consider parallel dynamics, i.e., agents are moving simultaneously in time steps of the length Δt. The adaptive time span will be introduced below. The movement consists of two parts, the decision process during which the agent chooses his desired target cell $\xi_i^*(t+\Delta t)$, and the conflict solution that may disable the agent to enter the desired cell ($\Rightarrow \xi_i(t + \Delta t) = \xi_i(t)$) or not ($\Rightarrow \xi_i(t + \Delta t) = \xi_i^*(t + \Delta t)$).

In the general concept, the decision process of agent i is driven by the hopping probability from x to y

$$p_i(x, y; t) = \Pr\left[\xi_i^*(t + \Delta t) = y \,|\, \xi_i(t) = x\, ,\, \eta\,(t; S_i(x))\right]\, , \qquad (1)$$

where the target cell y is chosen from the neighborhood $S_i(x)$, i.e., $p_i(x, y; t) = 0$ for $y \notin S_i(x)$; $\eta\,(t; S_i(x))$ stands for the state of the neighborhood. In this article, the translation and agent invariant Moore's neighborhood is used, i.e., $\forall i \in \hat{N}$ and $\forall x \in L, S_i(x) = (x + S_M) \cap L$, where the Moore's neighborhood of (0,0) is denoted $S_M = \{(-1,1);(0,1);(1,1);(-1,0);(1,0);(-1,-1);(0,-1);(1,-1)\}$.

To describe the conflict solution process, let us denote

$$\eta^*_x(t) = \Big\{ j \in \hat{N} \,|\, \xi^*_j(t) = x \Big\} \tag{2}$$

the set of agents "wishing" to enter x at time t. In the following the process is explained from the point of view of the agent i, therefore the notation $y^* = \xi^*_i(t + \Delta t)$ is used for convenience. If the target cell y^* is empty at time t, i.e.,$\eta_{y^*}(t) = 0$, two situations are distinguished according to the cardinality of $M := \eta^*_{y^*}(t + \Delta t) \subset \hat{N}$.

If $M = \{i\}$, the cell y^* is conflictless and $\xi_i(t+\Delta t) := \xi^*_i(t+\Delta t)$. Otherwise, another agents than i wishes to enter the cell y^* and conflict appears. With probability $\mu \in \langle 0, 1\rangle$ this conflict remains unresolved, i.e., $\forall j \in M, \xi_j(t+\Delta t) := \xi_j(t)$. With probability $1-\mu$ results the conflict to the motion of one agent from M; this agent is chosen randomly. The parameter μ plays the role of the friction parameter [5].

According to the presented concept of decision process (1) it is not excluded to chose an occupied cell as the target, i.e., if $\eta_{y^*}(t) = 1$, agent i creates the so called bound $(i \to y^*)$ to the cell y^*. The bound comes to life when the agent sitting in y^* moves at time $t+\Delta t$. This enables the motion of all agents bounded to y^* with conflict solution according to the situation of unoccupied target cell explained above. This rule is applied recursively to all bounds existing at $t+\Delta t$. This principle enables the motion in lines within one algorithm step, which is desired phenomenon of pedestrian flow.

So far, parallel updating scheme was considered. In the following a general concept will be presented, which we call the *adaptive time span*. There is a specific time sequence $(t_{i,n})_{n\in\mathbb{N}_0}$ unique for every agent. This sequence determines the moments when agent i is *activated* to actualize his position according to above mentioned rules. Looking at the system as a whole, the system state is actualized at times $(t_m)_{m\in\mathbb{N}_0}$, where

$$t_{m+1} := \min_{j\in\hat{N}}\{t_{j,n} \,|\, t_{j,n} > t_m \,,\, n \in \mathbb{N}_0\}\,. \tag{3}$$

This equations means that the time of next actualization is chosen as the nearest event defined by the updating sequences of all agents.

A common example of such concept is that the sequence $t_{i,n}$ is driven by the Poisson process, i.e., the increments $t_{i,n+1} - t_{i,n}$ are exponentially distributed. The principle used in our model comes out of the idea that each agent has its own desired frequency of actualization f_i characterized by the desired time increment $\tau_i = f_i^{-1}$. This leads, in an ideal case without correlations between agents, to the sequence $t_{j,n} = n\tau_j$. As will be shown below, the presented conflict solution changes this sequence slightly due to the adaptation to the movement of agents in the neighborhood. Similar idea is considered in [14] to simulate heterogenous system with two kinds of agents.

The concept of asynchronous updating scheme reduces the number of conflicts as well as the ability of holding in lines. To avoid the latter, following concept is introduced. Let us denote $A(t) = \{j \in \hat{N} \,|\, \exists n \in \mathbb{N}_0, t_{j,n} = t\}$ the set of active agents at time t. Let us consider the situation that agent i creates a bound $(i \to x)$ at $t_{i,n}$, because x is occupied by agent j, i.e., $\xi_j(t_{i,n}) = x$. As times evolves, two possibilities may occur. Firstly, agent j stays in x until $t_{i,n} + \tau_i$, i.e., $\xi_j(t) = x, \forall t \in (t_{i,n}, t_{i,n} + \tau_i)$. In this case, the bound $(i \to x)$ is canceled, the actualization time of i is set to $t_{i,n+1} := t_{i,n} + \tau_i$ and the decision process of i at $t_{i,n+1}$ goes according to above mentioned rules. Secondly, agent j moves from x within the time interval $(t_{i,n}, t_{i,n} + \tau_i)$. In that case we set

$$t_{i,n+1} := \min\{t \in (t_{i,n}, t_{i,n} + \tau_i) \,|\, \xi_j(t) \neq x\} \quad \text{and} \quad \xi_i^*(t_{i,n+1}) := x. \tag{4}$$

To complete the idea of adaptive time span the time penalization of diagonal movement should be mentioned. Because the distance between diagonally attached cells is $\sqrt{2}$ longer then the distance between vertically or horizontally attached cells, the transition to the diagonal cell should take adequately long time. To incorporate this feature to the model we propose the diagonal penalization in the sense, that after an agent moves diagonally at $t_{i,n}$ his next actualization time is set to $t_{i,n} + \tau_i \cdot 3/2$, where $3/2$ is the rational approximation of $\sqrt{2}$. The rational approximation is used to keep the agents partially synchronous in their updates to maintain conflicts. The requirement of partial synchronization leads to the idea, that agents are divided into small number of groups according to their updating frequency. We propose to divide the agents into two groups with desired time increments $\tau_1 = 1$ and $\tau_2 = 3/2$ which together with diagonal movement penalization leads to the time sequence of the whole system to be considered as $t_m = m \cdot 1/4$, see Fig. 2 for illustration.

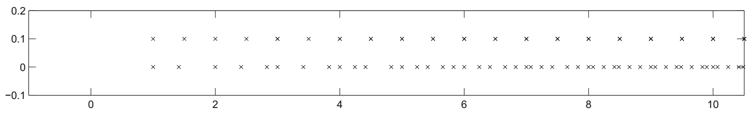


Fig. 2. Illustration of adaptive time span with the rational approximation of $\sqrt{2}$ (top) and without it (bottom).

The overall principle is illustrated by the example in Fig. 3.

3 Specification of the Decision Process $p_i(x, y; t)$

In this section the decision process $p_i(x, y; t)$ will be specified. For purposes of this article, slightly modified concept incorporating occupancy and movement prediction introduced in [1] is used.

Analogically to the Floor Field model, we define the potential field $U : L \to \mathbb{R}$ that assigns value $U(x)$ to every cell x in the sense that the agent is attracted

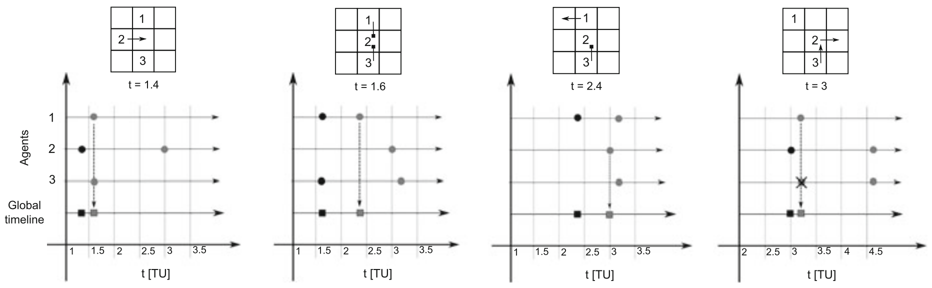


Fig. 3. Example illustrating the timeline with bounds. Agent 1 is faster ($f_1 = 0.8^{-1}$), agents 2 and 3 are two times slower, $f_{2,3} = 1.6^{-1}$. Bounds are indicated with squares, movement with arrows. This agent shows two situations: at $t = 2.4\ TU$ agent 1 cancels the bound, at $t = 3.0$ agent 3 uses the bound to synchronous movement with agent 2.

to the cell with lower potential. Commonly, the potential field is generated by the exit and is static over time. The probability of hopping from x to y fulfils $p_i(x, y; t) \propto \exp\{-k_U U(y)\}$, where $k_U \in \langle 0, 1\rangle$ stands for the parameter of sensitivity to the potential.

In the scope of this article, the decision process is influenced by the predicted state of the neighborhood $S_i(x)$. To describe this influence, let us first define the term movement prediction. In the essence, agent is predicted to move in the same direction as he did in his previous step, i.e, we define the predicted position of agent i for all $t \in (t_{i,n}, t_{i,n+1}\rangle$ as

$$\xi_i^P(t) := \begin{cases} 2\xi_i(t_{i,n}) - \xi_i(t_{i,n-1}) & \text{if } 2\xi_i(t_{i,n}) - \xi_i(t_{i,n-1}) \in L\,, \\ \xi_i(t_{i,n}) & \text{otherwise}\,. \end{cases} \tag{5}$$

This allows us to define the dynamical field $\eta_{x,i}^P(t)$ of the occupancy prediction of the cell x from the point of view of agent i as

$$\eta_{x,i}^P(t) := \begin{cases} 0 & \text{if } \{j \neq i \,|\, \xi_j^P(t) = x\} = \emptyset\,, \\ 1 & \text{otherwise}\,. \end{cases} \tag{6}$$

Now we can write the unnormalized weights $\tilde{p}_i(x, y; t)$ as

$$\tilde{p}_i(x, y; t) = d(x - y) \exp\{-k_U U(y)\} \big(1 - k_O \cdot \eta_y(t)\big)\big(1 - k_M \cdot \eta_{y,i}^P(t)\big)\,, \tag{7}$$

where $k_O, k_M \in \langle 0, 1\rangle$ are sensitivity parameters to the occupation and movement prediction. The symbol $d(x)$ represents the diagonal probability penalization that has to be incorporated because of the symmetry problems and is defined for $x \in S_M$ as $d(x) = c$ for $x_1 \cdot x_2 \neq 0$ (diagonal movement) and $d(x) = 1$ for $x_1 \cdot x_2 = 0$ (horizontal and vertical). The influence of above mentioned parameters is discussed in [1].

Contrarily to the basic Floor Field model we support the deterministic motion in free flow, i.e., in non congested regime. The agents decide stochastically according to (7) only if the cell with minimal potential $y_{\min} := \operatorname{argmin}\{U(y) \,|\, y \in S_i(x)\}$ is occupied or predicted to be occupied by another agent. This leads to the final form of decision process

$$p_i(x, y; t) = \begin{cases} \tilde{p}_i(x, y; t)/\mathcal{N} & \eta_{y_{\min}}(t) \cdot \eta^P_{y_{\min},i}(t) = 1\,, \\ \delta_{y_{\min},y} & \text{otherwise}\,, \end{cases} \tag{8}$$

where $\mathcal{N}$ is the normalization constant and $\delta_{i,j}$ is the Kroneker delta ($\delta_{aa} = 1$ and $\delta_{ab} = 0$ for $a \neq b$).

In simulations described below we have used the potential U in the form $U(x) = F \cdot \|x - e\|_2$, where $\|.\|_2$ is the euclidian norm and e is the position of the exit cell mostly set to $e = (0, 0)$. The constant F represents the strength of the potential and was set to $F = 3$ through the agreement of the simulations with experiments.

In the scope of this article, we will use the set of parameters determined through the egress experimental study mentioned in [1]. Specific values of the parameters are given in Table 1.

Table 1. Parameter values used for the simulation

Potential	Occupation	Movement	Friction	Diag. pen.	Time unit
k_U	k_O	k_M	μ	c	TU
1	0.2	0.7	0.9	0.2	0.31 s

Here we note that the friction parameter seems to be extremely high, but his influence is weakened by the adaptive time span and the concept of bounds and serves mostly as the means that enables the switches between lines in the chaotic queue in front of the exit. In the simulation we have considered the frequency to be measured in time units TU associated with the own frequency $f = 1$, i.e., The agent with own frequency $f_i = 1$ in free flow regime (without obstacles) moves one cell ahead during one time unit. Therefore one unit corresponds to 0.31 seconds to obtain realistic motion.

4 Experimental Study of the Phase Transition

The above mentioned model has been used to simulate the phase transition analogically to [2], where a situation of a rectangular room with one entrance and one exit has been introduced. The phase transition occurs due to the change of the inflow parameter α. In [2], the situation was studied by means of simple Floor Field model with agresivity parameter ζ. In this section, we present the experiment "passing through", which was inspired by the setting mentioned above.

The experiment was organized with help of 80 second year students of the Faculty of Nuclear Sciences and Physical Engineering of the Czech Technical

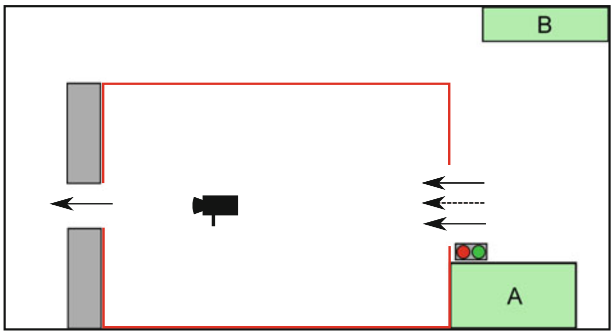


Fig. 4. The setting of considered experiment. The inflow was controlled by green signal A, pedestrians gathered at B. The situation was recorded by the camera placed above the room.

University. The room of 10 m × 6 m was arranged as depicted in Fig. 4 within the study hall of the faculty (Trojanova 13, Prague 2).

The inflow rate was controlled by the green signal on which a group of pedestrians (two or three) entered the room and were instructed to leave it through the exit placed at the opposite wall. Time intervals Δs between two successive signals were generated from trimmed Gaussian distribution with mean value μ_s which was varied from 1.43 s to 1.85 s. Because pedestrians were not able to react precisely on the signal while the mean value μ_s was set below 1.4 s, we have let more pedestrians to enter the room at the signal, what served us to increase the inflow rate.

Eight inflow settings have been considered. The setting is characterized by the number of entering pedestrians n_s and the mean value of time delay μ_s. From these the average inflow current J_{in} can be calculated as $J_{\text{in}} = n_s/\mu_s$ pedestrians per second. As we have used the time unit corresponding to 0.31 s as the algorithm step, we define $\alpha_{\text{exp}} = J_{\text{in}} \cdot 0.31$ pedestrians per model time unit (TU). This value gives the average inflow rate per time step, which is used for simulations. Therefore, α_{exp} corresponds to the probability of the injection of new pedestrian to the system within one TU. Experimentally measured values of α_{exp} are given in Table 2.

As seen from Table 2, the settings were classified according to macroscopic observations: free flow regime, congested regime, and meta-stable regime. In the free flow regime (settings 1, 2, and 3) pedestrians walked freely through the room and did not block each other in front of the bottleneck. In meta-stable regime, occasional conflict in front of the exit resulted to a small cluster formation, which melted after short time period (setting 4) or stabilized, but fluctuated around 5 pedestrians in the cluster (setting 5). By increasing the inflow rate, the competition at the door significantly blocks the motion and rapidly increases size of the stable cluster which fills in significant part of the room (settings 6, 7, and 8).

Table 2. Experimental setting: inflow rates

Setting	μ_s	n_s	J_{in}	α_{exp}	Observed state
1	1.78	2	1.12	0.35	free flow
2	1.68	2	1.19	0.37	free flow
3	1.59	2	1.26	0.39	free flow
4	1.43	2	1.40	0.43	temporary cluster
5	1.85	3	1.62	0.50	stable cluster
6	1.72	3	1.74	0.54	congestion
7	1.66	3	1.81	0.56	congestion
8	1.57	3	1.91	0.59	congestion

Analogical observation was made by the simulations using the model described in previous section. The transition from free regime to the stable congestion was observed at the inflow rate $\alpha_{\text{sim}} \in \langle 0.42, 0.46\rangle$ pedestrians per second, which corresponds to the experimental observation. At the critical values of α_{sim}, the creation and melting of the temporary cluster was highly supported by the asynchronous update and bounds principle.

Snapshots form the experiment together with the simulation realization are given in Fig. 5.

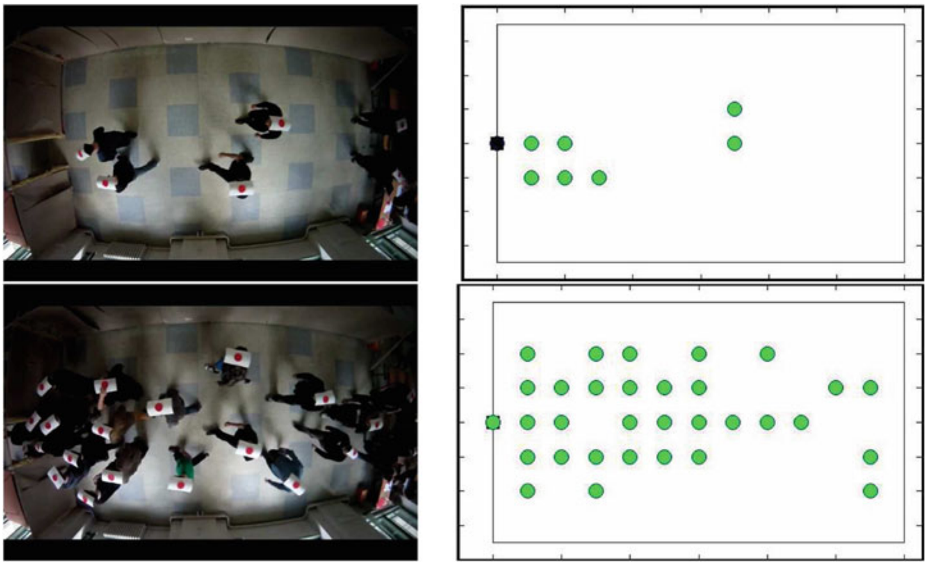


Fig. 5. Snapshots from the experiment (left) and model simulation (right) for low (upper) and high (lower) density.

5 Conclusion

A general concept of adaptive time span in cellular automata for pedestrian dynamics was introduced. Within this concept, the agents own actualization frequency f_i was presented together with the principle of bounds. This combination leads to the creation of chaotic queue near the bottleneck which is accompanied by the line formation and occasional switches between these lines. This improvement was inspired by the real behavior of pedestrians under experimental conditions.

Furthermore, the phase transition from low to high density was studied experimentally. As seen from the experiment and Table 2, the saturation of pedestrians at the bottleneck is closely related to the inflow rate in the critical area. Qualitative comparison of the model behavior and the experiment shows that the concept od bounds together with the movement and occupation prediction supports highly this wanted feature. Furthermore, due to this concept, the heterogenous system with more types of pedestrians can be implemented keeping the motion in lines intact.

Presented model has been qualitatively compared to the observed behavior and we believe that the modifications of the time span improves the agreement of the underlying Floor Field model with observed non panic behavior of pedestrians near the bottleneck.

Acknowledgements. This work was supported by the grant SGS12/197/OHK4/3T/14 and by the MSMT research program under the contract MSM 6840770039.

References

1. Hrabák, P., Bukáček, M., Krbálek, M.: Cellular model of room evacuation based on occupancy and movement prediction. J. Cell. Autom. **8**(5–6), 383–393 (2013)
2. Ezaki, T., Yanagisawa, D., Nishinari, K.: Analysis on a single segment of evacuation network. J. Cell. Atom. **8**(5–6), 347–359 (2013)
3. Klüpfel, H., Schreckenberg, M., Meyer-König, T.: Models for crowd movement and egress simulation. In: Hoogendoorn, S., Luding, S., Bovy, P., Schreckenberg, M., Wolf, D. (eds.) Traffic and Granular Flow 03, pp. 357–372. Springer, Berlin (2005)
4. Schadschneider, A., Chowdhury, D., Nishinari, K.: Stochastic Transport in Complex Systems: From Molecules to Vehicles. Elsevier Science B. V, Amsterdam (2010)
5. Kirchner, A., Schadschneider, A.: Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics. Phys. A Stat. Mech. App. **312**(12), 260–276 (2002)
6. Nishinari, K., Kirchner, A., Namazi, A., Schadschneider, A.: Extended floor field CA model for evacuation dynamics. IEICE Trans. Inf. Syst. **E–87D**, 726–732 (2004)
7. Kretz, T., Schreckenberg, M.: The F.A.S.T.-Model. In: El Yacoubi, S., Chopard, B., Bandini, S. (eds.) ACRI 2006. LNCS, vol. 4173, pp. 712–715. Springer, Heidelberg (2006)

8. Schadschneider, A., Seyfried, A.: Empirical results for pedestrian dynamics and their implications for cellular automata models. In: Timmermans, H. (ed.) Pedestrian Behavior - Models, Data Collection and Applications, pp. 27–43. Emerald Group, Bingley (2009)
9. Schultz, M., Lehmann, S., Fricke, H.: A discrete microscopic model for pedestrian dynamics to manage emergency situations in airport terminals. In: Waldau, N., Gattermann, P., Knoflacher, H., Schreckenberg, M. (eds.) Pedestrian and Evacuation Dynamics 2005, pp. 369–375. Springer, Berlin (2007)
10. Yamamoto, K., Kokubo, S., Nishinari, K.: Simulation for pedestrian dynamics by real-coded cellular automata (rca). Phys. A Stat. Mech. App. **379**(2), 654–660 (2007)
11. Kretz, T., Kaufman, M., Schreckenberg, M.: Counterflow Extension for the F.A.S.T.-Model. In: Umeo, H., Morishita, S., Nishinari, K., Komatsuzaki, T., Bandini, S. (eds.) ACRI 2008. LNCS, vol. 5191, pp. 555–558. Springer, Heidelberg (2008)
12. Steffen, B.: A modification of the social force model by foresight. In: Klingsch, W.W.F., Rogsch, C., Schadschneider, A., Schreckenberg, M. (eds.) Pedestrian and Evacuation Dynamics 2008, pp. 677–682. Springer, Berlin (2010)
13. Suma, Y., Yanagisawa, D., Nishinari, K.: Anticipation effect in pedestrian dynamics: modeling and experiments. Phys. A Stat. Mech. App. **391**(12), 248–263 (2012)
14. Weng, W.G., Chen, T., Yuan, H.Y., Fan, W.C.: Cellular automaton simulation of pedestrian counter flow with different walk velocities. Phys. Rev. E **74**, 036102 (2006)

The Use of GPGPU in Continuous and Discrete Models of Crowd Dynamics

Hubert Mróz, Jarosław Wąs(✉), and Paweł Topa

AGH University of Science and Technology,
al. Mickiewicza 30, 30-059 Kraków, Poland
{jarek,topa}@agh.edu.pl

Abstract. The aim of the study is twofold: firstly to compare the possibilities of using GPGPU (General-Purpose Computing on Graphics Processing Units) in continuous and discrete crowd dynamics simulation, secondly to draw conclusions on the applicability of GPUs in engines of professional crowd simulations. For this purpose the authors have implemented two models of pedestrian dynamics: continuous - Social Forces model and discrete, Cellular Automata based - Social Distances model. The presented simulations refer to outdoor, large area pedestrian movement.

Keywords: Crowd simulations · Crowd models · GPU · GPGPU

1 Introduction

The development of *Information and Communication Technologies* requires more and more often, reliable simulation of the crowd, operating in real time to support decision making process by managers responsible for the safety. On the other hand, an application of crowd simulation for entertainment purposes requires primarily high performance, although in recent years the emphasis is simultaneously located on realism of the simulation.

Using of graphic processing units GPU brings potential opportunity of development of reliable and efficient simulation. An idea of effective massive Agent Based modelling on the GPU was presented in [16] and it was suplemented by Navigation Vector Fields in [10]. A concept of real-time multi scaled simulation was presented in [21], while issues of real time crowd rendering was discussed in [14]. The use of GPUs for simulation of Social Force model combined with idea of toxic gases diffusion was described in [4]. Hybrid path planning during GPU based crowd simulations was recently proposed in [5], whilst [9] proposes using 3D data structure to improve performance of 3D rendering using GPU.

Based on bibliography and previous experience with crowd modeling, the authors decided to implement and test two popular models of crowd dynamics: continuous and discrete. The aim of the research is to assess the GPU technology for implementing continuous and discrete models of the crowd, with its focus on the simulation engine (not rendering). The article is a continuation of work by the authors [11].

Dedicated application was created using NVidia CUDA (Compute Unified Device Architecture) technology, and 3D Object-oriented Graphics Rendering Engine (OGRE 3D) environment was used for the visualization.

The document is organized as follows: GPU context of simulations is presented in Sect. 2, whilst two applied models: continuous and discrete are presented in Sect. 3. Next, results of simulations are described in Sect. 4 and concluding remarks are placed in Sect. 5.

2 Graphics Processing Units in Simulations

The computing power of Graphic Processing Units rapidly grown in recent years, significantly exceeding the performance achieved by top CPUs. This fact results in growing interest in using GPU for solving more general class of problems. Graphic processor shows its power only if the problem fits or it is adapted for this specific architecture. GPU programming uses the stream processing concept, where a set of instructions (called a kernel) is executed concurrently over all of the items of a dataset. Each instance of a kernel is called a thread. A set of threads are arranged into one-, two- or three-dimensional grid and such the group is called block. Threads from a single block are queued to execution automatically by processor's scheduler. Group of threads that are together scheduled to concurrent execution is called a warp. In order to achieve fully parallel execution, all threads in the warp need to do exactly the same operations. Otherwise, the threads will be scheduled to execution one by one. To prevent this situation, the programmer should avoid the conditional instructions that differentiate paths of execution.

GPU uses various types of memory. The fastest are registers but they have very small capacity (i.e. 32 kB per one streaming processor in Nvidia Fermi architecture). Shared memory is located on-chip and also provide high bandwith and low latency (about 1600 GB/s and 10–20 cycles in Fermi processors). Shared memory is accessible by all threads within a single block. Global memory has lowest bandwith and highest latency (about 400–800 cycles) and it is accessible by all threads. Proper use of different types of memory also is key to the efficient use of GPU computing power.

Graphic processors develop very fast. Subsequent hardware generations (marked as the Compute Capabilities of the device) usually have a larger amount of computational resources, but also their programming capabilities become more flexible and efficient.

The possibility of using GPU for solving more general problems appeared with introducing programmable shader units and shader programming languages as Nvidia Cg or GLSL [17]. Later, Nvidia introduces programming platform CUDA (Common Unified Device Architecture) with C for CUDA programming language. In 2008, the Khronos Group presented an open programming environment OpenCL (*Open Computing Language*) [13] dedicated to the same purposes.

In comparison to CUDA which is a proprietary solution and works only with Nvidia GPU, OpenCL is supported also by other manufacturers like AMD/ATI and Intel.

Models based on Cellular Automata paradigm can be efficiently implemented on GPU. Cells are usually organized into regular lattices and they can be simple mapped on grid of GPU threads. According to the definition of Cellular Automata, all cells must be updated in parallel by using the same set of rules. Thus, in natural way the CA rules can be encoded as kernels and each cell will be processed by a single thread. Amount of data that have to be exchanged between threads is limited due to locality of neighborhood.

Practical demonstration of usefulness of GPU for Cellular Automata modeling has been presented in many papers. Rybacki et al. [18] investigated seven different simulation strategies (two of them used GPUs) for Cellular Automata and compared them for a few well-known CA rules (e.g. Game of Life, Parity). He concludes their article with statement that the usefulness of GPU-based Cellular Automata algorithms strongly depends on the models that we want to simulate. Bilotta et al. [1] has ported MAGFLOW model [19] (lava flow simulation) to CUDA environment [1]. The algorithms were carefully optimized for GPU architecture as well as they took benefits from usage of shared memory. The authors evaluated performance for various graphic processors with different Compute Capabilities. The conclusions acknowledge that Cellular Automata models gives opportunity to be implemented on GPU with high efficiency, however the straightforward conversion from CPU to GPU can be disappointing. Also Topa et al. [20] investigated the possibilities of efficient implementation of the Cellular Automata model for water flow. Although they used only a global memory, the modification introduced in the algorithm provided high occupancy ratio. Due to regular structure of data the coalescing for transactions in global memory is achieved automatically and almost optimal. As the result, the GPU implementation appeared to be up to 100 faster than CPU version. Blecic et al. [2] used latest GPU with Nvidia Kepler architecture to implement model of urban dynamics. The original algorithm presented in this work also had to be modified to satisfy the requirements of GPU architecture. The authors also implemented additional version that uses shared memory and compared both approaches. The tests show that efficiency of GPU algorithms are 150 times better, then sequential CPU algorithm. The usage of shared memory gives only a little increase of performance due to hardware managed cache.

3 Implemented Models

3.1 Continuous Model - Social Force

The most popular model of pedestrian dynamics is the model proposed by Helbing and Molnar [8]. In Social Force Model pedestrians are represented as particles moving in a given direction and the particles are influenced by a set of forces: repulsion forces from walls and obstacles, repulsion or attraction forces from other pedestrians etc.

Thus, each of the agents/pedestrians is an independent object, that changes its state, based on the forces acting on him/her. These forces can be divided into three types: attractive force from target, forces from other pedestrians, repulsive force from obstacles. Agents react only to those agents who are located in their neighborhood. The neighborhood is determined by a distance and angle of view, and its allocated according to agent's movement direction.

In summary, Social Force model is a microscopic, force-based model, continuous in time and space.

In presented simulation world simulation is divided into so-called containers (sized $15\,\mathrm{m} \times 15\,\mathrm{m}$). Agents moving in space are assigned to a container. In the calculation of "state agent" only forces associated with the current container and neighbour containers are taken into account. Parallelization on GPUs is based on the simultaneous calculation of parameters of agents in different containers. Single thread performs calculations for agents located in a single container.

3.2 Discrete Model - Social Distances Model Based on Cellular Automata

Cellular Automata (CA) become more and more popular in many areas ranging from sociophysics [7] to granular flow modeling [15]. Such kind of discrete approach can be used in scheduling [6]. In area of crowd dynamics the framework of Cellular Automata is often combined with a concept of potential fields [3]. In these models a set of static and dynamic fields is applied, and the fields modify transition function of applied cellular automaton. The authors have applied such CA based floor field model with more precise representation of space, namely generalized Social Distances Model [22][1].

Pedestrians represented by ellipses, are allocated in a square lattice, where cells have a size $25\,\mathrm{cm} \times 25\,\mathrm{cm}$. The center of an ellipse coincides with the center of current cell [22]. Each cell can be occupied at the time by one agent in particular time step. The size of each ellipse equals $a = 22.5\,\mathrm{cm}$ - semimajor axis and $b = 13.5\,\mathrm{cm}$ - semiminor axis - which is assumed the average size of a person according to WHO data. Agent is allowed to move to the next cell in Moore neighborhood with radius 1 or to rotate by, at minimum, $45\,^{\circ}$. According to the chosen compressibility parameter $\epsilon \in \{0; 0.331\}$ - the maximum area of intersection of neighboring ellipses is assumed. If the move of a pedestrian on the specified field, does not cause overgrowth of value of compressibility parameter, then it is executed (the configuration of agents is specified as allowed). Pedestrians/agents move along the gradient of the potential to different, defined targets.

Fundamental diagram for Social Distances model was presented in [24]. The presented model is discrete in time and space, it is microscopic and rule-based.

[1] This model was originally designed as a hybrid: cellular automaton with a component of force, however in the study it is implemented exclusively as floor field, cellular automaton based model [23]

Table 1. Performance of rendering of 500 agents, environment size is 100 m × 100 m

Model	Full rendering	Level of details	Simplified rendering
Social Force	29 FPS	78 FPS	98 FPS
Social Distances	35 FPS	100 FPS	122 FPS

The basic structures for storing data in a model of Social Distances are two types of grids: grid of occupancy and static potential field [3]. Grid of occupancy is always only one, and stores information whether a cell is free, occupied by an obstacle or occupied by an agent. On the other hand, we can apply from one to several potential fields, depending on the complexity of the environment (number of agents' targets). An agent can change the potential fields, but at a particular time slice he/she can be associated with only one potential field. Parallelism of calculations in this model is based on the simultaneous processing of multiple agents (one thread is associated with a single agent). The thread retrieves information from the neighbourhood, it decides on the action and, if necessary, updates the information of the grid occupancy.

3.3 Technical Aspects of Implementation

The project has been implemented using Nvidia CUDA programming platforms. Applied technology enables the creation of executable code both on CPU and GPU, making it possible to compare the differences in performance. Contrary to many works on crowd modeling using the GPU, the main objective of the study is to investigate the effectiveness of mechanisms of the simulation.

Object-oriented Graphics Rendering Engine was used for rendering the graphical part of the simulation and it belonged to the additional tasks. Three scenarios were used for handling graphics: a full rendering, simplified rendering (Level of Details) and no rendering. Table 1 contains rendering performance for different methods. Unfortunately the rendering system appears to be insufficient in large population of agents. The authors are going to implement in future a dedicated rendering system that retrieves necessary information directly from the existing data blocks in the device memory.

Transferring data during simulation causes a performance bottleneck of the program, thus it is important to minimize the transfer as possible. For this purpose all data related to the simulation are sent only once to the device memory (after initialization). Since then, the system performs all computation calculations on a copy of the data. However, it is necessary to share information in the other side to enable actualization of graphical representation of the simulation. For this purpose, at every step of the simulation, minimal amount of data is transferred (for instance a vector with a current direction and a sense of pedestrians). These transfers will be unnecessary in dedicated rendering system. The structures of these data should satisfy the following assumptions: data must be easy to read both by the program running on CPU and GPU, data must be well packed, because they should occupy a minimal space. For these reasons, a simple data structures are proposed.

The simulation is carried out in three stages:

Initialization - the creation of objects - agents and representation of the environment, initializing data specific to the model (like potential fields), the creation of an appropriate calculation module and sending all necessary data to the device memory.

Updating - update calculation module, receive data from the device and update of the graphical representation.

Closuring - shutting down the simulation and remove all the initialized data.

One of the most important elements realized in GPU implementation is the use of containers for agents and for objects (for instance buildings) in their environment.

Decision-making algorithm for Social Distances model are strictly connected with potential fields implemented using GPU.

All the related data are stored in global memory. At this moment the original CPU algorithms were straightforward converted to CUDA platform. We intentionally decided not to introduce any optimizations in order to have a reference version and a starting point for further modifications.

4 Results

As a result of the implementation a flexible application was obtained. The application allows comparison of selected characteristics of continuous and discrete models of crowd dynamics based on both: CPU and GPU computing. Figure 1 shows the graphical user interface of the application. In the bottom left corner the current view on results of performance are visible - number of processed frames per second.

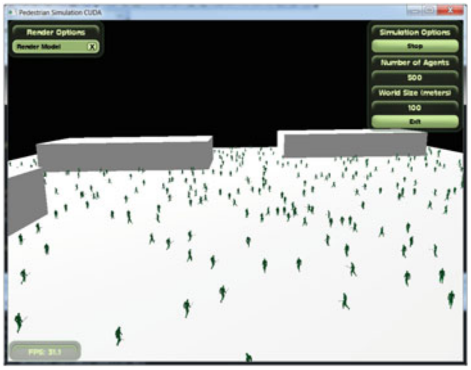


Fig. 1. Graphic user interface of the application

4.1 Performance Tests

In order to verify the efficiency of presented models, a series of tests were carried out. A criterion used for comparison purposes was the speed of the simulation, expressed in frames per second related to a number of agents. Computational effort and time required to perform the calculations in the simulation of crowd dynamics is closely related to the density of the population of agents. Therefore, tests were carried out in a similar population density for continuous and discrete models using CPU and GPU technologies.

Tests were carried out on the hardware platform equipped with: Dual-Core AMD Athlon II 250 (3.00 GHz), 4 GB of RAM and a graphics card GeForce GTS 250 with 512 MB RAM and Compute Capability 1.1. The tests described in the current section were performed with disabled rendering (because of the inefficient rendering module).

The chart (Fig. 2) shows the results of performance tests for models: Social Force and Social Distances carried out respectively on CPU and GPU. Performance in this case is expressed by the relationship: number of supported pedestrians to FPS (frame per second). As we can see on the chart for both models, simulations executed on GPU are characterized by a higher performance, than the corresponding CPU simulations. It is worthwhile emphasized that this increase is achieved for non-optimized algorithms.

As the discrete model, Social Distances in tests showed much greater efficiency then continuous Social Force: on CPU results are better by 160–300 %. By using GPU we are able to improve results by additional 60–70 %. As we mentioned before, we still have space for improvements by optimization of GPU

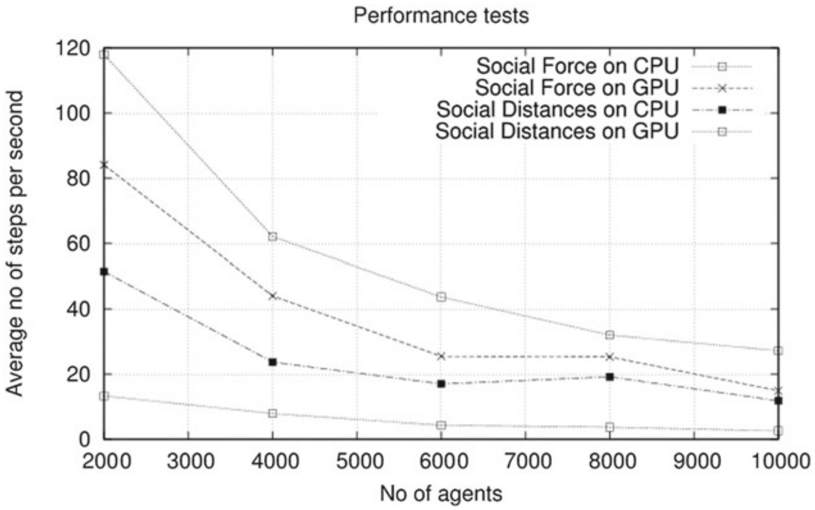


Fig. 2. Results of performance tests for Social Force and Social Distances model implemented respectively on CPU and GPU

algorithms. The increase of performance for discrete model was lower for GPU, because it requires a large amount of memory references. This is due to the fact that threads related to the agents must (during operation) access information about their neighbourhood, and then update the data in an occupancy field. This results in a memory overhead, because operations related to access to memory are very expensive (in a sense of calculation) and they may lead to bottlenecks in GPU applications. However, further optimizations can be achieved in this field. CUDA allows users to take advantage of different types of memory, such as global, shared or texture memory. All of them have pros and cons and appropriate usage can be crucial in application's performance [12].

5 Concluding Remarks

The authors present a comparison of two completely different models of crowd dynamics: continuous Social Force model and discrete, Cellular Automata based Social Distances model. Both models have been implemented using GPGPU and, for comparative purposes, using the CPU. It should be noted, that the use of the GPU requires a completely different approach to development of applications and in this case many limitations will appear (Sect. 2). Hence, popularity of using GPU in professional applications dedicated to the dynamics of the crowd, in particular in the simulation engines, is still limited.

Based on the analysis of many variants of the size and complexity of the environment and the available number of agents, authors proposed ranges of applicability of the two models created using the GPU technology (Sect. 4). Social Force model works well in large, sparsely populated areas, where the density of agents is relatively small. This continuous model works perfectly for free pedestrian traffic, where agents head for a large number of defined targets. Discrete Social Distances model gives good results for the simulation of objects with large number of obstacles and large population of agents. Due to the necessity of application of many potential fields, the GPU implementation of Social Distances model is suitable for simulation in which number of pedestrian targets is not high: it can be applied in evacuation scenarios or in relatively simple, freeway traffic scenarios. Social Force method is continuous, thus it allows for more accurate mapping of pedestrians movement. The trajectories of motion are much more accurate then in a discrete model. On the other hand, Social Distances method gives the possibilities of generation an environment with more complex topology (building, obstacle), whilst in Social Force model it is more problematic.

It should be emphasized that the development of GPU technology is rapid and it is expected that GPGPU will be more and more competitive, in relation to traditional CPU programming. During the implementation of the models presented in this work, the specification of CUDA technology has been changed several times and new features have been included (for example *Thrust* library of templated primitives). Currently, most of the professional applications dedicated to the dynamics of the crowd (especially those with engineering character) implement simulation engine using traditional CPU, but the immediate future may bring a change in this area.

Future works will concentrate on tuning the algorithms for efficient execution on GPU device. The profiler need to be used in order to identify the bottlenecks. The algorithm must be redesigned in order to increase the occupancy ratio. It is necessary to consider whether explicit usage of shared memory is profitable. Processors with Compute Capability 2.0 and higher (processors with Nvidia Fermi and Keppler architecture) have cache L2 for global memory transaction which may surpass the efficiency of manually managed shared memory.

Acknowledgment. This research is partially supported by FP7 project SOCIONICAL, No 231288.

References

1. Bilotta, G., Rustico, E., Hérault, A.: Porting and optimizing magflow on cuda. Ann. Geophys. **54**(5), 580–591 (2011)
2. Blecic, I., Cecchini, A., Trunfio, G.A.: Cellular automata simulation of urban dynamics through gpgpu. J. Supercomputing **65**, 614–629 (2013)
3. Burstedde, C., Klauck, K., Schadschneider, A., Zittartz, J.: Simulation of pedestrian dynamics using a two-dimensional cellular automaton. Phys. A Stat. Mech. App. **295**(3–4), 507–525 (2001)
4. Courty, N., Musse, S.R.: Simulation of large crowds in emergency situations including gaseous phenomena. In: Proceedings of the Computer Graphics International 2005, CGI '05, pp. 206–212. IEEE Computer Society, Washington, DC (2005)
5. Demeulemeester, A., Hollemeersch, C.-F., Mees, P., Pieters, B., Lambert, P., Van de Walle, R.: Hybrid path planning for massive crowd simulation on the GPU. In: Allbeck, J.M., Faloutsos, P. (eds.) MIG 2011. LNCS, vol. 7060, pp. 304–315. Springer, Heidelberg (2011)
6. Dudek-Dyduch, E., Kucharska, E.: Learning method for co-operation. In: Jędrzejowicz, P., Nguyen, N.T., Hoang, K. (eds.) ICCCI 2011, Part II. LNCS, vol. 6923, pp. 290–300. Springer, Heidelberg (2011)
7. Gwizdałła, T.M.: The dynamics of disproportionality index for cellular automata based sociophysical models. In: Sirakoulis, C.G., Bandini, S. (eds.) ACRI 2012. LNCS, vol. 7495, pp. 91–100. Springer, Heidelberg (2012)
8. Helbing, D., Molnár, P.: Social force model for pedestrian dynamics. Phys. Rev. E **51**, 4282–4286 (1995)
9. Joselli, M., Passos, E.B., Zamith, M., Clua, E., Montenegro, A., Feijo, B.: A neighborhood grid data structure for massive 3D crowd simulation on GPU. In: Brazilian Symposium on Games and Digital Entertainment, pp. 121–131 (2009)
10. Karmakham, T., Richmond, P., Romano, D.M.: Agent-based large scale simulation of pedestrians with adaptive realistic navigation vector fields. In: TPCG'10, pp. 67–74 (2010)
11. Mróz, H., Wąs, J.: Discrete vs continuous approach in crowd dynamics modelling using GPU computing. J. Cybern. Syst. **45**, 25–38 (2014)
12. NVIDIA. NVIDIA CUDA C Best Practices Guide (2010)
13. OpenCL. OpenCL - the open standard for parallel programming of heterogeneous systems

14. Peng, Ch., Park, S.I., Cao, Y., Tian, J.: A real-time system for crowd rendering: parallel LOD and texture-preserving approach on GPU. In: Allbeck, J.M., Faloutsos, P. (eds.) MIG 2011. LNCS, vol. 7060, pp. 27–38. Springer, Heidelberg (2011)
15. Płaczek, B.: A traffic model based on fuzzy cellular automata. J. Cell. Automata **8**, 261–282 (2013). (ISSN: 1557–5969)
16. Richmond, P., Romano, D.M.: Agent based GPU, a real-time 3D simulation and interactive visualisation framework for massive agent based modelling on the GPU. In: Proceedings of International Workshop on Supervisualisation 2008 (IWSV08), June 2008
17. Rumpf, M., Strzodka, R.: Graphics processor units: new prospects for parallel computing. Numer. Solut. Partial. Differ. Equ. Parallel Comput. **51**, 89–132 (2006)
18. Rybacki, S., Himmelspach, J., Uhrmacher, A.M.: Experiments with single core, multi-core, and GPU based computation of cellular automata. In: Conference on Advances in System Simulation, pp. 62–67 (2009)
19. Spataro, W., Rongo, R., Lupiano, V., Avolio, M.V., D'Ambrosio, D., Trunfio, G.A.: High detailed lava flows hazard maps by a cellular automata approach. In: Pina, N., Kacprzyk, J., Filipe, J. (eds.) Simulation & Modeling Methodologies, Technologies & Appli. AISC, vol. 197, pp. 85–100. Springer, Heidelberg (2013)
20. Topa, P., Młocek, P.: GPGPU implementation of cellular automata model of water flow. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011, Part I. LNCS, vol. 7203, pp. 630–639. Springer, Heidelberg (2012)
21. Vigueras, G., Orduña, J.M., Lozano, M.: A gpu-based multi-agent system for real-time simulations. In: Demazeau, Y., Dignum, F., Corchado, J.M., Pérez, J.B. (eds.) Advances in PAAMS. AISC, vol. 70, pp. 15–24. Springer, Heidelberg (2010)
22. Wąs, J., Gudowski, B., Matuszyk, P.J.: Social distances model of pedestrian dynamics. In: El Yacoubi, S., Chopard, B., Bandini, S. (eds.) ACRI 2006. LNCS, vol. 4173, pp. 492–501. Springer, Heidelberg (2006)
23. Wąs, J., Lubaś, R., Myśliwiec, W.: Proxemics in discrete simulation of evacuation. In: Sirakoulis, ChG, Bandini, S. (eds.) ACRI 2012. LNCS, vol. 7495, pp. 768–775. Springer, Heidelberg (2012)
24. Wąs, J., Myśliwiec, W., Lubaś, R.: Towards realistic modeling of crowd compressibility. In: Peacock, R.D., Kuligowski, E.D., Averill, J.D. (eds.) Pedestrian and Evacuation Dynamics, pp. 527–534. Springer, US (2011)

Modeling Behavioral Traits of Employees in a Workplace with Cellular Automata

Petros Saravakos and Georgios Ch. Sirakoulis(✉)

Department of Electrical and Computer Engineering,
Democritus University of Thrace, 67100 Xanthi, Greece
{psaravak,gsirak}@ee.duth.gr
http://gsirak.ee.duth.gr

Abstract. The aim of this paper is to examine a parameterized working environment on the basis of behavioral traits of employees in an organization. Firstly we define several behavioral traits of the employees, including the employee's attitude in the workplace, the influence radius and her/his reluctance to adapt to organizational norms, stated as insistence. The combination of these traits allows us to model employee interactions to a satisfactory extent for a realistic model of the working environment. Secondly, we define two metrics illustrating the policies adopted by the organization either to restrain unwanted or impose desirable behavioral patterns. Finally, the corresponding Cellular Automaton (CA) model enables us to utilize the aforementioned parameters and to simulate the under study workplace. The presented simulation results can be used as a complementary tool for managerial decisions illustrating workplace dynamics and forecast future trends.

Keywords: Behavioral Traits · Cellular Automata · Working Environment · Simulation

1 Introduction

The contemporary workplace is an ever-changing environment of high diversity stemming both from the management organization as well as the employees themselves. The workplace is a forum where a variety of different behaviors are expressed, each with a different consequence to the individuals within the organization as well as the entire organization [1]. While in the past decades there have been numerous studies on enhancing the efficiency of the managerial decisions with the intention of maximizing productivity, the main focus in past few years is been shifted to studying the impact of behavioral features of individuals or groups of individuals. Hence, terms such as work spirituality and employee behavior are regarded as key factors in promoting a company's goals i.e. productivity, customer service, minimization of costs, environmental consciousness. Changing the attitude of employees in the workplace is a twofold cause; encouraging positive behavior and alleviating negative behavior patterns, with main focus on the latter.

Positive behavior patterns consist of practices that might have a positive effect in relation to the company's goals, i.e. productivity, customer service, minimization of costs, environmental consciousness. Although positive behavior is considered to fall within the company lines, in some cases, positive behavior might diverge from the company guidelines. In this occasion, the employee behavior is asserted as positive deviance. According to Spreitzer and Sonenshein [11], positive deviance entails organizational citizenship behaviors, whistle-blowing, corporate social responsibility and creativity/innovation.

On the other hand, negative behavior patterns have more severe ramifications to a company. To a certain extent negative behavior is expected to occur and can be tolerated, given that the cost this practice entails is negligible. Nevertheless, negative deviant workplace behaviors should be abolished. Such behaviors include absenteeism, withdrawal, withholding effort and behaviors that lead to corporate inequality [9]. Negative deviance can be further distinguished based on its severity (minor – serious), and to its target (company – co-workers).

Consequently, organizations tend to restrict employee behavior by instigating policies that limit negative deviance. Such policies form organizational norms, a grouping of "expected behaviors, languages, principles and postulations that allow the workplace to perform at a suitable pace" [2]. In general, positive behavior is been acknowledged by the management and as thus is usually promoted by providing benefits, such as wage raise, promotion and fringe benefits. However organizational norms have not been as successful in restraining negative behavior and might rebound in setting restrictions to positive behaviors as well. A conventional précis of company policies leans toward a balance view of battling negative deviance, tolerating negative behavior, encouraging positive behavior and finally hesitant to positive deviance. Employee behavior has been a key issue in the fields of business management nonetheless, where most approaches focus on business processes modeling and simulation than on the employees themselves, and work psychology which is refrained to assessing and classifying employee behavior. We use the following example to illustrate our point; Let an employee A, who is very productive and is regarded an asset to the company, but at the same time has a negative deviance towards other employees. From a business management point of view, A has a remarkable performance while his co-employees negatively influenced by his behavior perform poorly. As a result they ought to be encouraged or disciplined to improve their productivity, regardless of the fact that the root of their bad performance is the behavior of A. From a work psychology point of view, it can be easily derived that A is the root of the problem and should be disciplined for his negative influence, but there are no modeling or simulation techniques to assess the business performance. From an engineering perspective, limited research is been conducted on simulations of employee behavior in the workplace. First, Hu and Zhang [5] proposed a model based simulation where the dynamics of employee incentive were modeled using Cellular Automata (CAs) [8]. The influence of employee behavior in the workplace has been modeled by Jiao et al. [6] who proposed a distance based behavior influence model using CAs.

In our study, we propose a CA model and examine the simulation results to study the influence of employee behavior in the workplace under the effect of company policies. Our model provides a comprehensive approach that illustrates the way a workplace evolves as the interactions of employees shape an ever-changing working environment as well as the impact of company policies to negotiate employee behavior. In contrast with the aforementioned models, our approach does not depend on distance-modeled influence but on the notion of several neighborhoods derived from a set of employee information. Even though the proposed model is not an end to itself in terms of employee management, it can be used to determine the robustness of the workplace and in tandem with applied marketing techniques to facilitate managerial decisions.

2 The Proposed CA Model

Cellular Automata (CAs) [8] are models of physical systems, where space and time are discrete and interactions are local. They have been extensively used as models for complex systems and have also been applied to several physical problems, where local interactions are involved [3,4,7,10,12,13]. A CA consists of a regular uniform n-dimensional lattice (or array), usually of infinite extent. At each site of the lattice (cell), a physical quantity takes on values. The value of this physical quantity over all the cells is the global state of the CA, whereas the value of this quantity at each site is its local state. A CA is characterized by five properties:

1. the number of spatial dimensions (n);
2. the width of each side of the array (w). w_j is the width of the j^{th} side of the array, where $j = 1, 2, 3, \ldots, n$;
3. the width of the neighborhood of the cell (r);
4. the states of the CA cells;
5. the CA rule, which is an arbitrary function F.

The state of a cell, at time step $(t + 1)$, is computed according to F, a function of the state of this cell at time step (t) and the states of the cells in its neighborhood at time step (t). For a 2-d CA, two neighborhoods are often considered: Von Neumann, which consists of a central cell and its four geographical neighbors north, west, south and east; and the Moore neighborhood contains, in addition, second nearest neighbors northeast, northwest, southeast and southwest, i.e. nine cells. In most practical applications, when simulating a CA rule, it is impossible to deal with an infinite lattice. The system must be finite and have boundaries, resulting to various types of boundary conditions such as periodic (or cyclic), fixed, adiabatic or reflection.

In our approach, we model the employee behavior in the workplace using a $m \times n$ CA array, where $m \times n$ is equal to the total number of employees. Furthermore, we assume that each employee has several features, reflected by the state of each cell. These features include persistent traits, that do not change

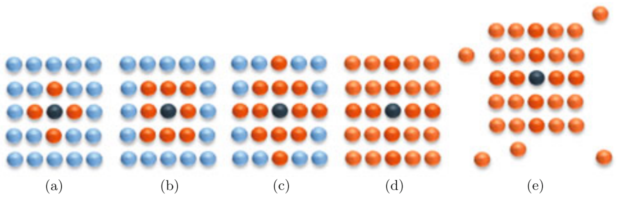


Fig. 1. Neighborhoods defined by different influence radii. (a) Von Neumann neighborhood of 5 cells, (b) Moore neighborhood of 9 cells, (c) diamond-shaped neighborhood of 13 cells, (d) Moore neighborhood of 25 cells, (e) basic Moore neighborhood enriched with a random number of random non-adjacent cells.

through time and adaptable, non-persistent, traits, which are affected as the employee adapts to the workplace.

Employee behavior is assessed on a seven point scale, hence each cell can take seven different values -can be in seven separate states- with regards to employee behavior ranging from -3 to $+3$. The negative (positive) states denote negative (positive) behavior and specifically, $-3(+3)$ denotes strong negative (positive) deviance, $-2(+2)$ negative (positive) deviance, and $-1(+1)$ negative (positive) behavior. The zero value state corresponds to the neutral behavior, thus a behavior following the company's organizational norms. Employee behavior can be assessed based on purely work-related criteria, such as performance evaluations, in tandem with personality traits that affect the employee's work behavior, such as organizational, leadership or motivational skills, derived by personnel evaluations. It is evident that the employee behavior adjusts as he is been influenced by his co-workers.

Secondly, each cell has an influence radius depicting the ability of the employee to influence his coworkers. The influence radius illustrates the locality of employee interactions and as such shapes the CA neighborhood. The influence radius can be given five values, ranging from 1 to 5, responding to each neighborhood shown in Fig. 1. In case the influence radius is equal to five, then the neighborhood expands to non-adjacent cells with the intention to illustrate of the employee to influence employees not in his vicinity, i.e. employees in different departments of a company. The influence radius is considered to be a persistent trait of the employee, portraying his ability to interact with his coworkers and does not change as the CA rule is been applied.

Each cell has another state, depicting each employee's insistence, his ability to remain uninfluenced by his coworkers. Insistence takes values from 1 to 5, where:

– 1 denotes that an employee is highly influenced by his neighborhood and thus his behavior is determined by his coworkers. As a result, the impact of the state of the central cell can reduced by a factor α_1 or ignored altogether.
– 2 denotes that an employee is mildly influenced by his neighborhood and thus his behavior is determined by his coworkers. As a result, the impact of the state of the central cell can reduced by a factor α_2 or ignored altogether.

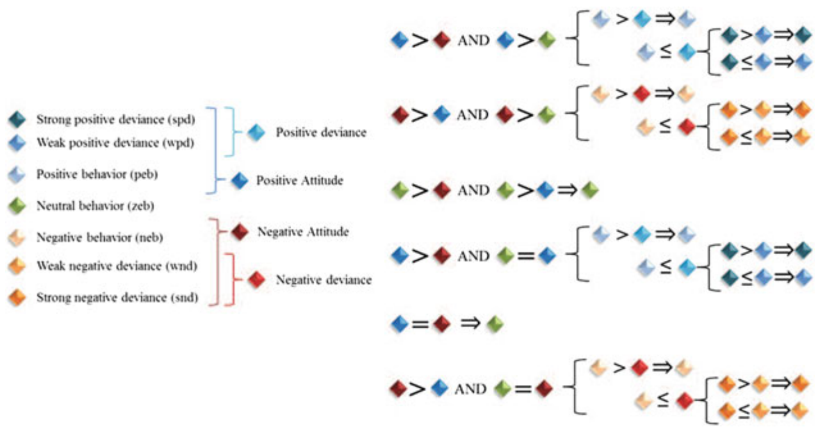


Fig. 2. Symbolic representation of the rules applied to the CA at each iteration.

- 3 denotes that an employee is influenced to a reasonable extend by his coworkers.
- 4 denotes that an employee is mildly reluctant to his coworkers' influence. As a result the impact of the states of the neighboring cells can be reduced by a factor α_3.
- 5 denotes that an employee is reluctant to his coworkers' influence. As a result the impact of the states of the neighboring cells can be reduced by a factor α_4.

Unlike influence radius, insistence is an adaptable trait and depends on the extent the employee is conformed to the organizational norms. Organizational norms are determined by the company policy. As mentioned before, the managerial decisions (limit positive deviance, tolerate mild negative deviance and eradicate strong negative deviance) cannot influence the employee's behavior, but only offer motives and deterrents to persuade the employee to renegotiate his stance and adapt to the workplace, hence to the dominant behavior of his coworkers.

Lastly, a set of rules has to be determined to estimate the change of employee behavior as well as of the adaptable features at each time step. The rules are uniformly applied to the entire CA and are summarized in the Fig. 2. These graphical rules are simple cases taking into account the aforementioned behavioral traits and their interaction as explained before. For example the first graphical rule declares that if (spd+wpd+peb)>(npd+wnd+neb) AND (spd+wpd+peb) >zeb then if peb>spd+wpd it means that out=peb, else if spd>wpd then out=spd, or else out=wpd. The same notation applies for the rest five rules trying to cover the examined behaviours.

After having described the CA model for our problem, we proceed with the corresponding simulation results, taking into account the following model parameters: the dimensions m and n of the $m \times n$ CA array; the number of iterations, i.e. the number of times the CA update rule is applied; the Company Reward Policy (CRP) which represents the company's intervention in order to keep employees within the organizational norms (in our case, the CRP is indicated as a number between 0 and 1, though it is usually restrained in the range [0, 0.5]); the Company Penalty Policy (CPP) which represents the company's intervention in order to force employees to conform to the organizational norms (as before, the CPP is indicated as a number between 0 and 1, though it is usually restrained in the [0, 0.5]); finally, the maximum number of extended neighbors, which denotes the maximum number of non-adjacent cells taken into consideration when the central cells has influence radius equal to five.

The selection of an appropriate company policy, reward or penalty, is crucial to reassess the insistence of each employee and thus must be chosen carefully. Apparently, the values of the aforementioned parameters depend greatly on the managerial decisions of the company and specifically whether a company is more interested in offering motives for conformation to the company policies or in discouraging deviance. A motivating technique will increase the percentage of conformed employees but will not have any particular impact on deviant behavior. On the contrary, a discouraging technique will reduce deviant behaviors but will not offer incentives to positive and most importantly negatively inclined employees to fully conform to the company policy. In our approach we assume only two metrics summarizing the company policies, the company reward policy and the company penalty policy. The company reward policy is applied only to employees with conformed and tolerable, negative or positive, behavior. In contrast, the company penalty policy is applied to the rest of the employees with different factors depending both on their behavior and their insistence. For instance, the company penalty policy is enhanced by an additional factor for strong negative deviant behaviors and by a smaller factor for strong positive ones, since the former have a more detrimental effect on the workplace. In both cases the above metrics are regarded and applied as weights in conjunction to the insistence of the examined CA cell. Moreover, regarding the initialization of the values of employee behavior, influence and insistence, all three variables are supposed to follow a Gaussian distribution in the appropriate range of values.

In our approach we introduce a new combined statistical metric, used to estimate the conformation rate of the employee to the company's organizational norm. This metric is referenced as employee loyalty or simply loyalty. Loyalty is a combined statistic of the employee behavior and his insistence. Employees with behavior closer to the employee norm and high insistence are considered very highly loyal, whereas employees with strong deviance -positive or negative- are considered very highly disloyal, since it is most unlikely that they might conform to the organizational norms. The direct consequences of the application of company policies on employee behavior are depicted on the loyalty graph than in any other graph.

We will examine three typical cases of workplaces to illustrate the simulation of employee behavior with CAs. Namely, the cases we will examine include: (1) A small-medium enterprise (SME), (2) a large company with minor company

Table 1. 1st, 2nd and 3rd case model parameters

Model parameters	(Case 1)	(Case 2)	(Case 3)
Number of employees	100 (10 × 10)	2500 (50 × 50)	2500 (50 × 50)
Number of iterations	6	15	15
Company Reward Policy	0 %	20 %	40 %
Company Penalty Policy	10 %	30 %	50 %
Number of extended neighbors	10	10	10

intervention and (3) a large company with significant company intervention. In the first study case, we will examine the model of a SME, a company model common in most developed countries of the Eurozone. SMEs have a limited number of employees, yet are renown of promoting innovation with reduced interest in applying strict company policies. Hence we can determine the model parameters as shown in Table 1. The results of the simulation ranging from hotter colors for positive to cooler colors for negative behavior, are depicted in Fig. 3, respectively. Given that the company it is not within the company's interest to conform its employees to a strict organizational norm, deviances are dominant compared to normal or conformed behaviors. Moreover the conservative penalty policy leads to lower insistences and as seen in Fig. 3, results in the slight descent of the strong negative deviance and at the same time in faster ascent of the strong positive deviance. Nevertheless, the loyalty statistics do present high conformation and discourage disloyalty.

In the second case, the model of a larger company will be taken into consideration. The increased size of the company usually imposes the need of managerial decisions to restrain deviant behaviors up to a certain extent, shifting the focus from creativity and innovation to a more efficient and smoothly operating working environment. This case can be further divided into two separate studies of company intervention, the stringency of the company policies. The model parameters are also presented in the Table 1 and have been selected in such a way to depict more clearly the resulting differences on company intervention. The results of the simulation for both cases can be illustrated in Figs. 4 and 5, respectively. In both cases, we note that the conformity of the employees is guaranteed, but most importantly in the latter case, where the company intervention was more intense, the employees are slightly less loyal but the neutral behavior in the employee behavior graph is increased more rapidly and reaches a higher value, exceeding the strong negative deviance. The insistence, a parameter directly affected by the company policy, also changes in a different manner than before. Despite the fact that low insistence is still dominant, employees attain much higher levels of very high insistence. Specifically, in first case, the mild company intervention results in the existence of strongly deviant behaviors in the workplace. Due to the local effect of the influence of each and every employee in our case, strong deviances create clusters, groups of people behaving in a similar fashion. The formation of such clusters is not condoned by the company policies but the mild company

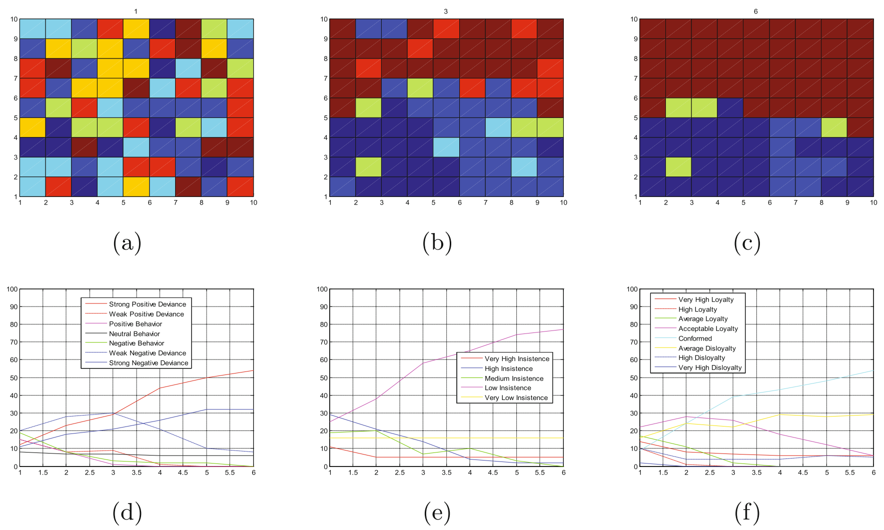


Fig. 3. Evolution of the employee behavior in the CA array of a SME (1st case) for different iterations, i.e. (a) $i = 1$, (b) $i = 3$ and (c) $i = 6$. Graphical representation of simulation statistics of the first case for (d) Employee behavior, (e) Insistence and (f) Loyalty, respectively.

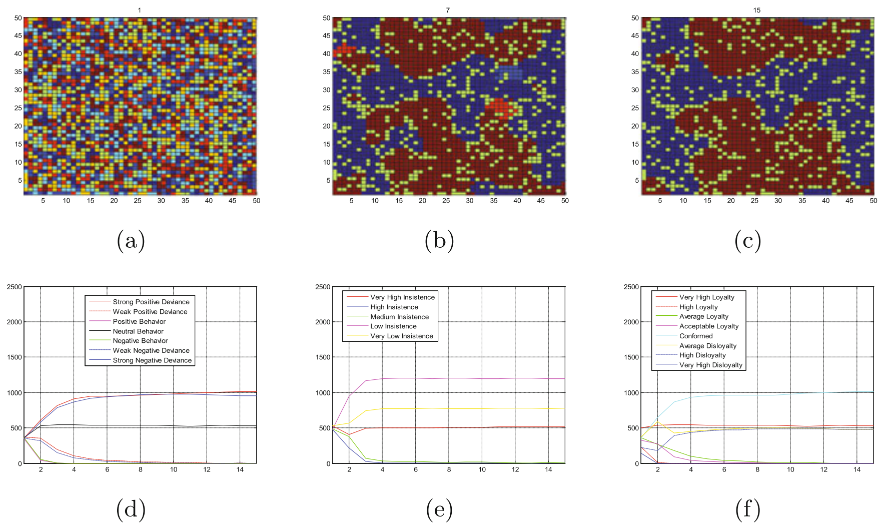


Fig. 4. Evolution of the employee behavior in the CA array of a large company (2nd case) for 15 different iterations. Snapshots of (a) $i = 1$, (b) $i = 7$ and (c) $i = 15$ are provided in correspondence. Moreover simulation statistics of (d) Employee behavior, (e) Insistence and (f) Loyalty for model's parameters in 2nd case are also shown.

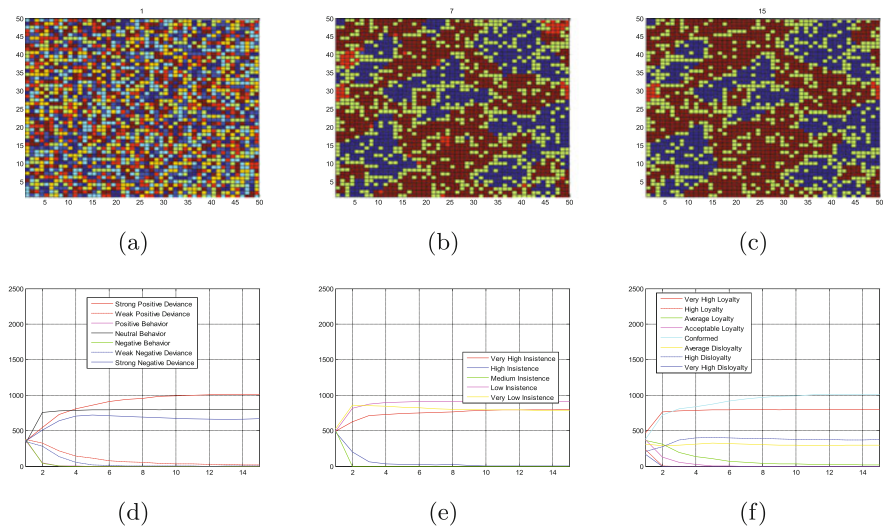


Fig. 5. Evolution of the employee behavior in the CA array of a large company (3^{rd} case) for 15 different iterations. Snapshots of (a) $i = 1$, (b) $i = 7$ and (c) $i = 15$ are provided in correspondence. Moreover simulation statistics of (d) Employee behavior, (e) Insistence and (f) Loyalty for model's parameters in 3^{rd} case are also shown.

intervention on altering the employee behavior cannot allow them to be disparaged or even isolated, in the time interval specified in this study case. Lastly in second case, the more intervening company guidelines are effectively reducing deviances and most importantly clusters of deviant behavior are of a significantly smaller size, since employees with neutral (conformed) behavior traits are scattered across the behavioral map. Therefore, it is safe to conclude that the effect of company polices is not actually focused on eliminating deviances but to discourage the formation of clusters of deviant behaviors.

3 Conclusions

In this paper we propose a CA model to simulate a workplace with regards to employee behavioral traits and in accordance with a variable company policy. The impact of the influence of inter-employee interactions as well as the importance of an appropriate company policy have been demonstrated in several study cases and depicted by statistical measures as the employee behavior, influence radius, insistence and the conformity measure, stated as employee loyalty. Conclusively, the CA model facilitates the presentation and simulation of a workplace with a variety of employee behavioral characteristics and under adaptable company policies. The proposed model can be practically used on two levels, firstly to estimate the workplace robustness and secondly to illustrate workspace dynamics. As a result it can be employed in conjunction with applied employee management techniques to facilitate managerial decisions and forecast the impact of employee behavioral changes and company decisions. As future work concerns, we are going to utilize our model to simulate behavioral patterns at a small enterprise, existent organization with mild company intervention, in Greece.

References

1. Appelbaum, S., Iaconi, G., Matousek, A.: Positive and negative deviant workplace behaviors: causes, impacts, and solutions. Corp. Gov. **7**, 586–598 (2007)
2. Coccia, C.: Avoiding a "toxic" organization. Nurs. Manage. **29**(5), 32–33 (1998)
3. D'Ambrosio, D., Spataro, W.: Parallel evolutionary modelling of geological processes. Parallel Comput. **33**, 186–212 (2007)
4. Georgoudas, I., Sirakoulis, G.C., Scordilis, E., Andreadis, I.: A cellular automaton simulation tool for modelling seismicity in the region of Xanthi. Environ. Model. Softw. **22**(10), 1455–1464 (2007)
5. Hu, B., Zhang, D.: Distance based cellular automata simulation for employee behaviors. Syst. Eng. Theory Prac. **2**, 83–96 (2006)
6. Jiao, Y., Sun, S., Sun, X.: Simulation of employee behavior based on cellular automata model. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007, Part IV. LNCS, vol. 4490, pp. 134–137. Springer, Heidelberg (2007)
7. Maerivoet, S., Moor, B.D.: Cellular automata models of road traffic. Phys. Rep. **419**(1), 1–64 (2005)
8. von Neumann, J.: Theory of Self-Reproducing Automata. University of Illnois Press, Urbana (1966)
9. Robinson, S., Bennett, R.: A typology of deviant workplace behaviors: a multidimensional scaling study. Acad. Manag. J. **38**(5), 555–572 (1995)
10. Sirakoulis, G.C., Bandini, S. (eds.): ACRI 2012. LNCS, vol. 7495. Springer, Heidelberg (2012)
11. Spreitzer, G.M., Sonenshein, S.: Toward the construct definition of positive deviance. Am. Behav. Sci. **47**(6), 828–847 (2004)
12. Topa, P.: Dynamically reorganising vascular networks modelled using cellular automata approach. In: Umeo, H., Morishita, S., Nishinari, K., Komatsuzaki, T., Bandini, S. (eds.) ACRI 2008. LNCS, vol. 5191, pp. 494–499. Springer, Heidelberg (2008)
13. Wąs, J., Gudowski, B., Matuszyk, P.J.: Social distances model of pedestrian dynamics. In: El Yacoubi, S., Chopard, B., Bandini, S. (eds.) ACRI 2006. LNCS, vol. 4173, pp. 492–501. Springer, Heidelberg (2006)

Probabilistic Pharmaceutical Modelling: A Comparison Between Synchronous and Asynchronous Cellular Automata

Marija Bezbradica(✉), Heather J. Ruskin, and Martin Crane

School of Computing, Centre for Scientific Research and Complex Systems Modelling (Sci-Sym), Dublin City University, Dublin, Ireland
{mbezbradica,hruskin,mcrane}@computing.dcu.ie

Abstract. The field of pharmaceutical modelling has, in recent years, benefited from using probabilistic methods based on cellular automata, which seek to overcome some of the limitations of differential equation based models. By modelling discrete structural element interactions instead, these are able to provide data quality adequate for the early design phases in drug modelling. In relevant literature, both synchronous (CA) and asynchronous (ACA) types of automata have been used, without analysing their comparative impact on the model outputs. In this paper, we compare several variations of probabilistic CA and ACA update algorithms for building models of complex systems used in controlled drug delivery, analysing the advantages and disadvantages related to different modelling scenarios. Choosing the appropriate update mechanism, besides having an impact on the perceived realism of the simulation, also has practical benefits on the applicability of different model parallelisation algorithms and their performance when used in large-scale simulation contexts.

Keywords: Discrete systems · Controlled drug delivery systems · Complex modelling · Parallel algorithms

1 Introduction

Probabilistic models based on Monte Carlo and CA frameworks have emerged in recent years as a viable response to the modelling needs imposed by design requirements of novel, more complex, drug delivery systems (DDS) [1].

Unlike traditional, differential equation based models, [2,3], CA attempt to recreate system-level behaviour by *in silico* simulations of individual interactions within the modelled device. This fits naturally with the early stages of the design process, in which global physico-chemical behaviours of DDS are investigated. By providing a low-cost alternative to lengthy, and potentially expensive, *in vitro* experiments, probabilistic computational modelling becomes an integral part of the drug design process. Nevertheless, uncertainties are inherent in this approach to modelling physical phenomena and parameters can multiply rapidly, due to the many different physical interactions within the model, so good understanding of model design choices is crucial.

In research papers covering the application of CA to the field, both synchronous and asynchronous update methods have been used [3–6], without going into deeper analysis of pros and cons of each. Choosing the algorithm for iterating through the cellular matrix and the order of application of the local rules affects how temporal realism of the physical process is represented. As DDS is biological in nature, chaotic or random updates might represent the system dynamics better than synchronous, "all-at-once", changes. On the other hand, as size and complexity of the models grows, the need for efficient parallelisation of model space restricts the application of the asynchronous methods due to performance reasons [7]. Therefore, it is of importance to understand the effects of synchronicity and asynchronicity to the model outputs, in order to be able to make optimal choices during model development. The transition of CA to ACA in general has been investigated in literature in a number of modelling contexts [7–9].

In this paper we compare behavioural characteristics, model outputs and performance for different synchronous and asynchronous CA update mechanisms in the context of probabilistic models used in controlled drug delivery and their parallel implementation, where differential equations are not applicable due to inherent unknowns in the parameter space. Finally, we analyse the results obtained by running the resulting models for a specific case of *coated drug bead formulations* [10].

In what follows, Sect. 2 presents the design methodology used for developing the CA rule sets, along with comparison of different CA and ACA update mechanisms when used in the context of the model and gives a theoretical analysis of their properties and variations in parallel and sequential implementations. Section 3 describes the developed model, with analysis of obtained results in Sect. 4, followed by the final discussion (Sect. 5).

2 CA and ACA Modelling

2.1 Design Methodology

As for any model build, the first stage involves transfer of domain knowledge of structural and behavioural characteristics of the DDS to the CA model. There are several distinct DDS characteristic categories to be considered:

- The shape and geometry of the system (slab, cylindrical or spherical) - captured in the shape and size of the model cellular matrix;
- Polymer composition of the device defined by states of the matrix cells;
- Polymer physico-chemical interaction mechanisms (laws) - described by characteristic behaviours that occur inside the DDS;
- Drug loading and initial dispersion within the device;
- Influence of the dissolution environment on polymer behaviour.

The models thus obtained, with the above characteristics, are classified as *kinetic* CA or ACA models [11]. Based on the way we choose to represent the physical phenomena modelled, we can adopt rules, either deterministic or probabilistic, (or a combination of both) affecting individual cell behaviour and the surrounding neighbourhood.

Although it is common for various families of CA update mechanisms to operate under periodic boundary conditions [12,13], pharmaceutical models benefit from using the fixed equivalent, as drug movement across the matrix boundaries is used as a direct method for calculating release rates. To satisfy the condition that the models need to mimic the non-homogeneousness of the physical device, with exact distributions of polymer and drug properties not available from experimental data, the model establishes the initial cell states using stochastic distributions within the known device geometry. Therefore, various direct Monte Carlo algorithms provide a natural solution to the initial condition problem, by establishing a random starting state for each simulation run.

2.2 Update Methods

Crucially, a good description of the model dynamics, i.e. the closest qualitative match to the behaviour of the pharmaceutical system modelled, relies on choosing the appropriate update method of the cellular automaton itself. This is addressed here, with particular emphasis on the correctness of the update rules as we consider several standard synchronous and asynchronous CA update methods [11,14,15].

Mathematically, the principal features of the 3-dimensional DDS models can be represented as a cellular automaton by a tuple representation, as given by:

$$ACA = \{G, A, U, \Theta, F\} \tag{1}$$

where G denotes a set of cell coordinates (the model matrix in our meta model). In the case of a 3D system:

$$G = Z^3 = \{(i, j, k) \mid 1 \leq i \leq N_x, 1 \leq j \leq N_y, 1 \leq k \leq N_z\} \tag{2}$$

A is the model alphabet - a finite set of possible cell states, (aggregate polymer states in the conceptual model), and U denotes the cell neighbourhood (including cell itself). Then *A(U(x), t)* denotes the state of a neighbourhood of cells U around a given cell x at a moment in time t. The behaviour of the system is described by a set of elementary transition rules for the conceptual model, F, where these are applied to the states of a neighbourhood of cells U. For the sequential case of synchronous updates, the general form of the rules (F_s) can be written as following:

$$F_s = \{f(x) : A(U(x), t) \rightarrow A(U(x), t + 1) \mid x \in G\} \tag{3}$$

Finally, Θ denotes the CA/ACA update order function, applied to G and A in order to advance the global model state. As a basis for Θ we investigated the application of several random and ordered asynchronous update methods, (see e.g. [16]) and compared these to the well-used synchronous method.

We implement the different update forms as modifications of the basic **synchronous CA two-phase update algorithm** of the main matrix G. The first ACA update method investigated is the **random order algorithm** which involves updating cells of G in a random order which is changed every time a full cell sweep is finished. All cells are updated in each time step of the simulation. The **random cyclic algorithm** is a variation of the random order algorithm, with the difference that a single random permutation of G is always used. Random permutation of G is chosen at the beginning of the simulation. In the **random independent** method one cell is chosen at random for updating at each time step. In the overall simulation, each cell should thus be updated approximately the same amount of times. However, over shorter time periods a given cell may be updated significantly more often. To achieve uniform selection, the algorithm thus depends heavily on the size of the sequence of the random number generator implementation. The *Mersenne Twister* algorithm has been used in this case, to reduce bias [17]. Finally, the **fixed cyclic sequential algorithm** was used in two forms: in the first one, cells of G are visited in sequential order of their coordinates (first width, then depth, then height in 3D). In the second form, cells are sorted based on their state (A), so that polymers of certain type are given simulation priority over polymers of other types (outer coating, then the inner coating, then the core). The order of cell simulation within the same polymer type is sorted by its coordinates.

2.3 Equivalence of Sequential and Parallel Implementations

In a concrete, model execution context, the mechanisms presented above only apply as long as the simulation is *sequential*. Once the algorithm has to scale up to be applicable to large data sets, the inclusion of parallelisation will have fundamental impact on the update logic.

It can be shown that in our case synchronous matrix updates are more suitable to parallelisation, as the effect of parallel updates on the resulting state should be equivalent. Consider a parallel version (F_p) of the fundamental rule set given in Eq. 3:

$$F_p = \{f(x_1, \ldots, x_n) : \bigcup_{i=1}^{n} A(U(x_i), t) \rightarrow \bigcup_{i=1}^{n} A(U(x_i), t+1) \,|\, x_1, \ldots, x_n \in G\} \quad (4)$$

Essentially, parallelising the update mechanism by splitting the CA space into disjoint domains, each having a set of boundary cells, introduces a simultaneous update of n cells at a time, where n represents the degree of parallelisation. The exact selection of cells $x_1, \ldots, x_n$ depends on the particular parallel algorithm being used. In the synchronous case, the state of a neighbourhood of cells $U(x)$ at moment t only depends on the same state for the previous moment $t-1$, and not on any currently updated state of any of the other neighbourhoods.

Therefore, for synchronous updates, it holds that $F_S \Leftrightarrow F_P$, which is in line with [18].

For asynchronous updates, the equivalence of sequential and parallel implementation breaks down. As the parallel version of the rule set presents a composition of functions applied simultaneously, the order of their application can result in a different overall state of the matrix. This is always true if any of the chosen neighbourhoods $U(x_i)$ overlap.

Therefore, in the case of random order and random cyclic updates, we expect the parallelisation to always result in slightly different model output. The same holds for different variations of *fixed cyclic rules*, where the idea of the underlying algorithm essentially breaks down. The only exception to this rule is the *random independent* order of updates, which might produce equivalent results, but only if, in each individual iteration, cells x_i are chosen in each parallel domain so that their neighbourhoods $U(x_i)$ are non-overlapping.

From an implementation point of view, when implementing parallelisation of pharmaceutical models using some of the industry standard parallel APIs, such as Message Passing Interface (MPI), synchronous updates are preferable from the execution speed point of view, as simulations have a practical wall-time limit of 24 h, the amount of time it would take to run a single *in vitro* experiment. Synchronous updates are extremely efficient in terms of execution speed especially as they can utilise two-sided communication using MPI to send and receive primitives. Asynchronous parallelisation schemes have to utilise one-sided communication primitives such as MPI "put" and "get", utilising the remote memory access mechanism, which, although slower, allows for the cell state to be asked for or provided on demand, without the need to wait on some eventual update [19].

Finally, it is important to note here that according to [20], for relatively slow changing stochastic CA models, the expected variance in outputs between synchronous and asynchronous update methods would be small. This results from the fact that large-scale, low-probability models do not have too many cell state updates in each iteration, which in turn limits the number of cases where overlapping neighbourhoods are updated.

3 CA Model for Coated Drug Formulations

Table 1 outlines the main CA rules used in the resulting model for each of the simulated processes. Following the notation from Sect. 2.2, each of the transition functions is applied to an alphabet of CA states:

$$A = \{P_{COAT}, P_{CORE}, PW_{COAT}, PW_{CORE}, B, D\} \tag{5}$$

where P_{COAT}, PW_{COAT}, P_{CORE} and PW_{CORE} denote the coating layers and core polymers, and their wetted state, respectively. B represents buffer cells and D drug molecules. The rules affecting each cell type can be described using a formal notation:

Table 1. Cell types and rules of behaviour for the examined model

Cell type	Behaviour description
Buffer (B)	Acts as a perfect sink for drug dissolution; Rules: diffusion; dissolution.
Coating polymer (P_{COAT})	Protective coating layer. Upon water penetration erodes into (PW_{COAT}); Rules: erosion; Initial state: assigned random lifetime using Erlang distribution.
Core polymer (P_{CORE})	Binds drug in the solid phase. Upon water penetration erodes into (PW_{CORE}); Rules: erosion.
Wet coating polymer (PW_{COAT})	Coating layer with some water penetration through the polymeric chains, allowing drug to diffuse. Applicable rules: diffusion.
Wet core polymer (PW_{CORE})	Result of core erosion allowing drug diffusion through relaxed chains; Rules: erosion; diffusion; swelling.
Drug packet (D)	Agent, initially dispersed in core polymer cells. Each cell can hold a maximum (saturation) amount of drug "packets". Initial distribution of packets throughout the sphere is determined using MC methods.

- **Erosion:** Polymer lifetime of a given cell x ($l(x)$) decreases linearly with time according to the following function: $f_e(x) : \{l(x) \rightarrow l(x) - t \mid \forall x \in \{P_{COAT}, PW_{COAT}, P_{CORE}, PW_{CORE}\}\}$
- **Diffusion:** The amount of drug present in cell x_a ($d(x_a)$) partially transitions to a neighbouring cell x_b with probability p_{diff}: $f_{diff}(x_a, x_b) : \{d(x_a, x_b) \xrightarrow{p_{diff}} d(x_a) - \Delta d, d(x_b) + \Delta d \mid \forall x_a \in \{D\}, x_b \in U(x_a)\}$
- **Swelling:** The amount of polymer present in cell is distributed in a similar fashion, using probability p_s: $f_s(x_a, x_b) : \{l(x_a, x_b) \xrightarrow{p_s} l(x_a) - \Delta l, l(x_b) + \Delta l \mid \forall x_a \in \{PW_{CORE}\}, x_b \in U(x_a), x_b \in \{P_{COAT}, P_{CORE}, B\}\}$
- **Dissolution:** Finally, the process of partial or total drug dissolution is described as the reduction in drug molecule count of a given cell once it transitions to solvent state: $f_{diss}(x) : \{d(x) \xrightarrow{p_d} d(x) - n \mid \forall x \in \{B\}, n \leq d(x)\}$

Multiple rule combinations can be superimposed (e.g. $f(x) = f_e(f_{diff}(f_s(x)))$) to fully define a cell behaviour during a single iteration if the given cell state satisfies all the alphabet preconditions of the rules given in Eq. 5.

4 Experimental Results

For each of the described update mechanisms, simulations investigated the following:

- The shape of the release curve during a 24 h period (a characteristic of GI tract transition time for the drug);

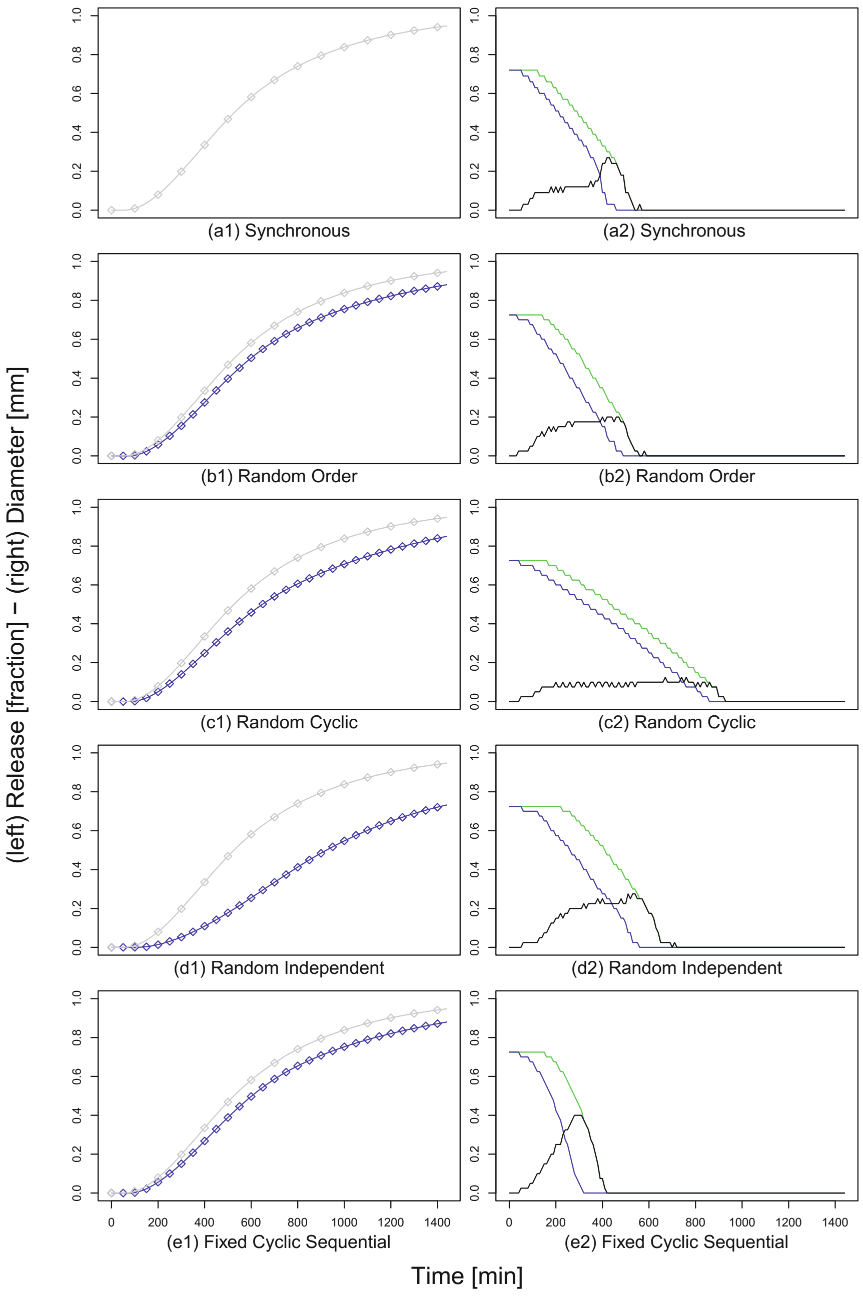


Fig. 1. 24 h simulation period for different update methods. **Left** - Drug release curves (blue - release fraction for appropriate update mechanism, gray - synchronous reference); **Right** - dissolution fronts changes over time (green swelling front, blue - erosion front, black gel layer thickness) (Color figure online)

- The radii of two main reaction fronts: the swelling front, (where the polymer moves from being in a static to dynamic state), and the erosion front (where polymer starts dissolving). These are the best indicators of the spatial scope of the underlying phenomena, which cannot be observed directly from release data alone;
- The device composite structure changes during characteristic stages of the dissolution, through visualisation of details.

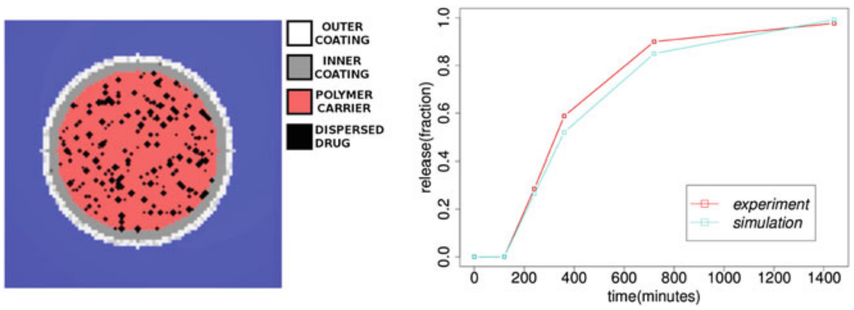


Fig. 2. Left- modelled device schematics; **Right** - An example experimental vs. simulated results for the case of synchronous updates (experimental data provided by Sigmoid Pharma Ltd) (Color figure online)

In Fig. 1 we show the results for *synchronous* updates (used as a basis for relative comparison with all subsequent ACA methods). The presented data is considered stable, as variations between different runs of the same parameter set were negligible. Comparing with *random order updates*, (Fig. 1b), we find a good match, with negligible release curve difference (indicating that the methods are effectively interchangeable e.g. where synchronous update is deemed more appropriate (for specified structure for example [21])). By comparing the curves analytically using the f_2 similarity factor [22], we obtain the results ranging from 50.93 (10 % fit) for random independent to 77.83 (3 % fit) for random order type of updates. However, random order, random cyclic and fixed cyclic independent have very similar f_2 values (3 %–4 %) fit so looking at the release curves alone is not enough to establish a clear advantage of one over another.

Figure 1 shows possible alternatives to the asynchronous *random order method*. It can be seen that *random independent* selection (Fig. 1d) produces release curves, which are significantly shifted with respect to those expected, although the radii behaviour is similar, in the sense that polymer transitions occur at the same rate. The features which give rise to this discrepancy can be observed in the model visualisations, where large *drug clusters* (black diamonds) occur as a consequence of some cells being updated more often than others (Fig. 3d). *Random cyclic updating*, on the other hand, produces release curves which are qualitatively similar to those expected, (Figs. 1(c1), (c2) and

3(c)), although the radii decrease dynamics are much slower. Finally, Figs. 1(e1), (e2) and 3(e), show results obtained using sequential matrix sweeps. This approach is not recommended due to the significant *bias*, which can be observed in the visualisation, leading to highly unrealistic radii dynamics. As expected, since the stochastic model is both slowly changing, with usual probability values used are much smaller than 1, and highly symmetrical, due to the spherical device geometry, the results obtained do not show highly anomalous results as would be expected in general CA to ACA transition [23], in line with theorems presented in [20] relating to equivalences between slow changing synchronous and asynchronous update processes in CA models.

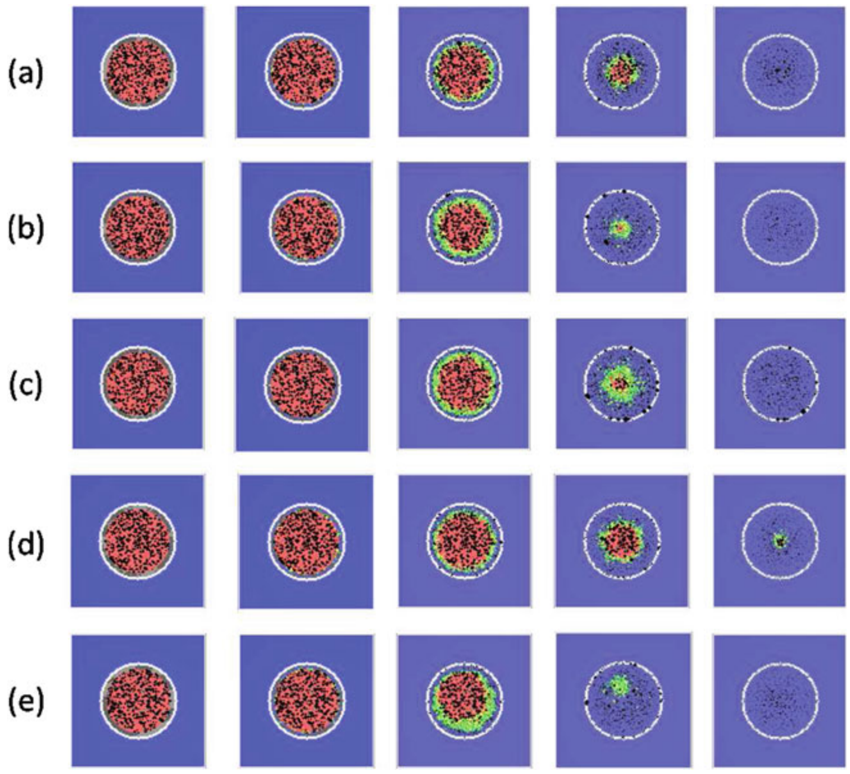
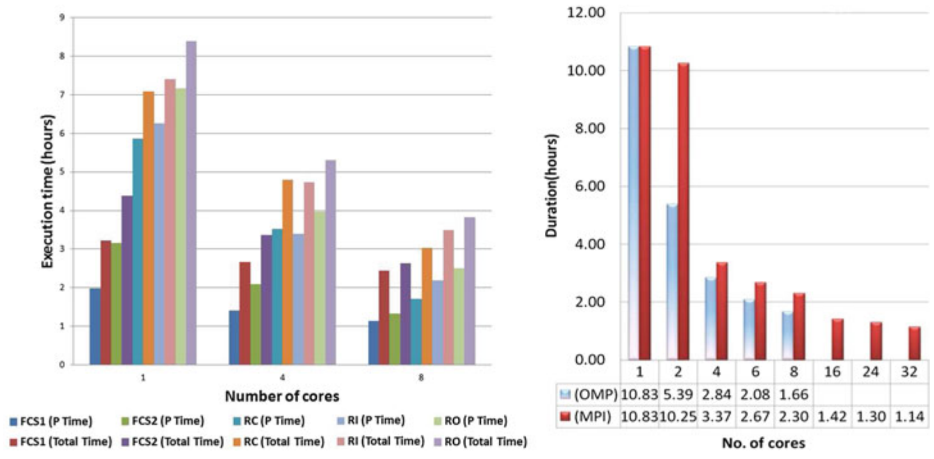


Fig. 3. Model visualisation during 10, 30, 150, 400 and 700 min interval, respectively: (a) synchronous; (b) random order; (c) random cyclic; (d) random independent; (e) fixed cyclic sequential. (Color figure online)

The validity of the synchronous updates when compared against experimental data is shown in Fig. 2, with obtained similarity factor showing a match within the standard variability range (<6 %).

At the end, we examine the overall performance in terms of simulation length using different ACA mechanisms in thread-level parallelisation context.



	1	2	4	6	8	16	24	32
(OMP)	10.83	5.39	2.84	2.08	1.66			
(MPI)	10.83	10.25	3.37	2.67	2.30	1.42	1.30	1.14

Fig. 4. Comparison of parallel and total simulation times for different synchronous and asynchronous update mechanisms. **Left** - comparison of different ACA update methods using thread-level parallelism. **Right** - comparison of synchronous update mechanism for thread-level (blue) vs. process-level (red) (Color figure online)

As expected, sequential algorithms were the fastest, as these could utilise the CPU memory cache better. The performance is closely followed by random cyclic variants, which might make an optimal choice for the scenario where the best simulation performance is needed, as opposed to state update realism, considering the accuracy of model outputs presented earlier. The worst performing are the random order algorithms which are not able to leverage the processor cache due to constantly changing order of memory access. However, these offer the best simulation realism and precision when compared to the synchronous variant, so the pros and cons of each should be weighed when making the decision. Figure 4 shows the performance profile for synchronous updates when switching from thread-level to process-level parallelisation model [21]. Although synchronous updates do not perform on the same level as asynchrounous ones, they do not have a parallelisation limit, and thus, ultimately, can be scaled to any number of nodes allowed by the model size.

5 Conclusions and Future Work

We have presented an overview of methodological considerations, important to modelling drug delivery systems using CA and ACA, and have analysed advantages and disadvantages of each update method. While some flexibility is possible in choosing between asynchronous and synchronous methods for approximate solutions in this context, this is governed by structural requirements. Our findings show that one of the most adequate solutions is *random order asynchronous*. In this regard, model visualisation provides valuable additional insight on structural behaviour and dissolution mechanisms, which is not readily apparent from working with standard release curve data alone, or which are intractable to supplementary experiment. The findings are useful for future modelling scenarios where it may be necessary to switch from one update mechanism to the other, both in terms of large-scale optimisation, but also in response to the need for describing component interactions in tailored solutions for individualised treatment. The CA pharmaceutical models presented here are a step in that direction. Conclusions drawn in this paper can also be applied in general to any slow chaining CA system, such as those used in social behaviour modelling for example (especially the performance part).

Acknowledgments. Financial support from the ERA-Net Complexity Project, P07217, is warmly acknowledged. The parallel simulations discussed in this paper were performed on computational resources provided by Sci-Sym, DCU, and by the Irish Centre for High-End Computing.

References

1. Haddish-Berhane, N., Jeong, S.H., Haghighi, K., Park, K.: Modeling film-coat non-uniformity in polymer coated pellets: a stochastic approach. Int. J. Pharm. **323**(1–2), 64–71 (2006)
2. Kaunisto, E., Marucci, M., Borgquist, P., Axelsson, A.: Mechanistic modelling of drug release from polymer-coated and swelling and dissolving polymer matrix systems. Int. J. Pharm. **418**(1), 54–77 (2011)
3. Siepmann, J., Siepmann, F.: Mathematical modeling of drug delivery. Int. J. Pharm. **364**(2), 328–343 (2008)
4. Barat, A., Crane, M., Ruskin, H.J.: Quantitative multi-agent models for simulating protein release from PLGA bioerodible nano- and microspheres. J. Pharm. Biomed. Anal. **48**(2), 361–368 (2008)
5. Göpferich, A.: Bioerodible implants with programmable drug release. J. Controlled Release **44**(2–3), 271–281 (1997)
6. Laaksonen, H., Hirvonen, J., Laaksonen, T.: Cellular automata model for swelling-controlled drug release. Int. J. Pharm. **380**(1–2), 25–32 (2009)
7. Bandman, O.: Synchronous versus asynchronous cellular automata for simulating nano-systems kinetics. Bull. Novosib. Comput. Cent. Ser. Comput. Sci. **25**, 1–12 (2006)
8. Alba, E., Giacobini, M., Tomassini, M., Romero, S.: Comparing synchronous and asynchronous cellular genetic algorithms. In: Merelo Guervós, J.J., Adamidis, P.A., Beyer, H.-G., Schwefel, H.-P., Fernández-Villacañas, J.L. (eds.) PPSN VII. LNCS, vol. 2439, pp. 601–610. Springer, Heidelberg (2002)
9. Kalgin, K.V.: Parallel simulation of asynchronous cellular automata evolution. Bull. Novosib. Comput. Cent. Ser. Comput. Sci. **27**, 55–62 (2008)
10. Bezbradica, M., Ruskin, H.J., Crane, M.: Modelling drug coatings: a parallel cellular automata model of ethylcellulose-coated microspheres. In: Proceedings of the International Conference on Bioscience, Biochemistry and Bioinformatics (ICBBB 2011), vol. 5, pp. 419–424 (2011)
11. Bandini, S., Bonomi, A., Vizzari, G.: What do we mean by asynchronous CA? A reflection on types and effects of asynchronicity. In: Bandini, S., Manzoni, S., Umeo, H., Vizzari, G. (eds.) ACRI 2010. LNCS, vol. 6350, pp. 385–394. Springer, Heidelberg (2010)

12. Burstedde, C., Klauck, K., Schadschneider, S., Zittartz, J.: Simulation of pedestrian dynamics using a two-dimensional cellular automaton. Physica A **295**(3–4), 507–525 (2001)
13. Xiao, X., Shao, S., Ding, Y., Huang, Z., Chen, X., Chou, K.C.: Using cellular automata to generate image representation for biological sequences. Amino Acids **28**, 29–35 (2005)
14. Baetens, J.M., der Weeën, P.V., Baets, B.D.: Effect of asynchronous updating on the stability of cellular automata. Chaos, Soliton. Fract. **45**(4), 383–394 (2012)
15. Schoenfisch, B., de Roos, A.: Synchronous and asynchronous updating in cellular automata. Biosystems **51**(3), 123–143 (1999)
16. Cornforth, D., Green, D.G., Newth, D.: Ordered asynchronous processes in multi-agent systems. Physica D **204**(1–2), 70–82 (2005)
17. Matsumoto, M., Nishimura, T.: Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. **8**, 3–30 (1998)
18. Bandman, O.: Parallel simulation of asynchronous cellular automata evolution. In: El Yacoubi, S., Chopard, B., Bandini, S. (eds.) ACRI 2006. LNCS, vol. 4173, pp. 41–47. Springer, Heidelberg (2006)
19. Thakur, R., Gropp, W.D., Toonen, B.: Minimizing synchronization overhead in the implementation of MPI one-sided communication. In: Kranzlmüller, D., Kacsuk, P., Dongarra, J. (eds.) EuroPVM/MPI 2004. LNCS, vol. 3241, pp. 57–67. Springer, Heidelberg (2004)
20. Toffoli, T., Margolus, N.: Cellular Automata Machines: A New Environment for Modeling. MIT Press, Cambridge (1987)
21. Bezbradica, M., Crane, M., Ruskin, H.J.: Parallelisation strategies for large scale cellular automata frameworks in pharmaceutical modelling. In: 2012 International Conference on High Performance Computing and Simulation (HPCS), pp. 223–230 (2012)
22. Moore, J.W., Flanner, H.H.: Mathematical comparison of curves with an emphasis on in-vitro dissolution profiles. Pharm. Technol. **20**(6), 64–74 (1996)
23. Fatès, N., Thierry, E., Morvan, M., Schabanel, N.: Fully asynchronous behavior of double-quiescent elementary cellular automata. Theoret. Comput. Sci. **362**(1–3), 1–16 (2006)

The Graph of Cellular Automata Applied for Modelling Tumour Induced Angiogenesis

Paweł Topa(✉)

Department of Computer Science, AGH University of Science and Technology,
al. Mickiewicza 30, 30-059 Kraków, Poland
topa@agh.edu.pl

Abstract. Angiogenesis is the process of formation of vascular network. Blocking tumour induced angiogenesis is one of the treatments applied in oncology. Research involving computer simulations looking for the rules influencing the structure of vascular network and its functionality. This paper summarizes the applications of Graph of Cellular Automata modelling tool, developed by the Author, for modelling Tumour Induced Angiogenesis. Vascular network which is modelled by the graph interacts with surrounding tissue represented by the lattice of automata. The network is developed and reorganized accordingly to locally acting factors (stimulators and inhibitors). The model includes blood flow calculations in a modelled vascular network.

Keywords: Cellular automata · Tumour-induced angiogenesis · Complex systems

1 Introduction

Vascular network is formed during embryogenesis and later in adulthood this process is quiescent and rigorously controlled by mutual interactions of various stimulators and inhibitors. Abnormal angiogenesis is triggered by development of solid tumours [1]. Therapies targeted against Tumour Induced Angiogenesis are subjects of wide scientific interdisciplinary investigations [2,3].

Without blood vessels, tumour cannot grow beyond a critical size and/or invade other regions of a body. The tumour induced angiogenesis starts when the production of pro-angiogenic factors overcomes other forces that kept the angiogenesis quiescent so far. Oxygen and nutrients penetrate the tissue only at a certain distance from the blood vessel. More distant cells subjected to metabolic stress synthesize many angiogenic stimulators (Tumour Angiogenic Factors — TAFs) among which the most famous is VEGF (Vascular Endothelial Growth Factor) [1]. Stimulators migrate towards the nearest blood vessels. When they reach the blood vessels, endothelial cells lining the vessel wall are activated. They start to proliferate and migrate towards the tumour cell attracted by the stimulators. The wall of the parent blood vessel becomes degraded, and it opens to form a new capillary. Migrating and proliferating cells form a hollow tube-like cavity (the lumen), which are stabilised later by smooth muscle cells and pericytes [4].

In this paper, a computer model developed for modelling tumour induced angiogenesis is presented. The model exploits computational paradigm of Cellular Automata. However, modifications we introduced in order to reflect phenomena that make up the process of forming vascular network, make that the data structures and the algorithm are far more complex than classical Cellular Automata models. Our tool called "Graph of Cellular Automata" was previously applied for modelling evolution of river systems [5]. After minor modification, it can be applied for simulation angiogenesis [6,7].

The paper is organized as follows. Section 2 briefly discusses several existing cellular automata models of tumour induced angiogenesis. In Sect. 3, the model, its design and implementation is discussed. Sample simulation results are presented in the following section. The article ends with a summary.

2 Existing Models of Tumour Induced Angiogenesis

Tumour Induced Angiogenesis is modelled by using the continuous approach: Partial Differential Equations – PDE, i.e., [8,9] and the discrete approach: Cellular Automata, Cellular Pots model, etc. [10,11]. Also hybrid models which mix continuum and discreet simulations were proposed many times [12,13].

Continuous models usually employ Partial Differential Equations in order to reflect distributions of various angiogenic factors and endothelial cells. Modelling of this nature can predict the behaviour of certain averaged quantities such as vessel and tip densities per volume tissue, but is unable to provide the details of microscopic features such as vessel length or structure of network.

Discrete approach assumes that the modelled particles (cells or molecules of chemical substances) are tracked individually. Their behaviour is controlled by a specific set of biophysical rules. This approach is particularly useful for studying the dynamics of individual cells, as well as various types of cell-cell interactions and cell-factor. Rule-based approach allows for easy translation of specific biological processes into a set of algorithmic instructions for each cell. On the other hand, the computational cost of simulation depends on the number of particles used and the number of steps of the simulation. Although the rule-based approach is a convenient way of describing complex biophysical processes, the verification and calibration of such the models can be non-trivial.

Stokes and Lauffenburger [14] put the hypothesis that the shape of the blood vessel is determined by the trajectory of the moving active cell located at the tip of the vessel. This tip migrates with velocity v in a continuous two-dimensional domain, attracted by source of angiogenic factors. Anderson and Chaplain [11] also assumed that the growth of the single vessel is governed by the move of the endothelial cell located at the sprout tip. This cell moves across the regular rectangular lattice according to defined rules. At each step of the simulation cell moves in one of the four directions or remains in place with a certain probability.

The probabilities are calculated by using the continuous approach, i.e. the diffusion equation supplied with terms reflecting the influence of angiogenic factors. Thus their models can be treated as hybrid [15].

Cancer growth and tumour induced angiogenesis are the phenomena involving many inter-related processes across a wide range of spatial and temporal scales. Multi-scale models evolved from hybrid models that, in fact, possessed at least two levels, i.e. tissue and vascular. Macklin et al. [13] introduced a multi-scale model that directly based on Anderson and Chaplain works. The model included blood flow module and capillary adaptation and remodelling module. Owen et al. [16] developed a model of tumour angiogenesis, which involves several processes: the growth of blood and blood flow, oxygen diffusion, the development of normal and neoplastic cells. Multi-scale model is implemented through the introducing several layers, which directly correspond to the biological processes that occurs in different spatio-temporal scales.

3 The Model of Tumour-Induced Angiogenesis

The approach used in the model presented in this paper separates computational models of the vascular network and the surrounding tissue. Figure 1 presents spatio-temporal dependencies between the physical processes simulated by these components. Blood flow calculation refers to process that is relatively fast and covers whole modelled network. Production of angiogenic factors and their diffusion occurs within a few seconds and take place in the scale of intra-and inter-cellular. Developing a network of blood vessels is slower and lasts for hours and days, and the spatial scale ranges from intra- to extra-cellular.

Figure 1 also demonstrates dependencies between computational components within the model. Arrows present how the components communicate and influence each other. Connection means that state(or states) calculated by module are treated as an input for the module indicated by the arrow.

The foundations of the model presented here are as follows:

- Tumour cells do not migrate nor proliferate.
- “Hungry” tumour cells produce TAFs.
- Vascular sprouts grow toward the increasing concentrations of TAF.
- The rate of maturation depends on local concentration of pericytes.
- Nutrients and oxygen are supplied only by vessels that transport blood.
- Vessels that do not transport blood undergo gradual disintegration.
- Antiangiogenic Factors (AAFs) prevent vessels from forming or maturing.
- We use reflective boundary conditions – a sprout that reaches edge of the mesh changes the direction of its movement to the opposite.

The model can be defined in a formal way as:

$$CA_{ANG} =< Z^2, G_{CA}, X_K, S, \delta >, \quad \text{where :}$$

- Z^2 — a collection of cells ordered as a square or hexagonal grid,

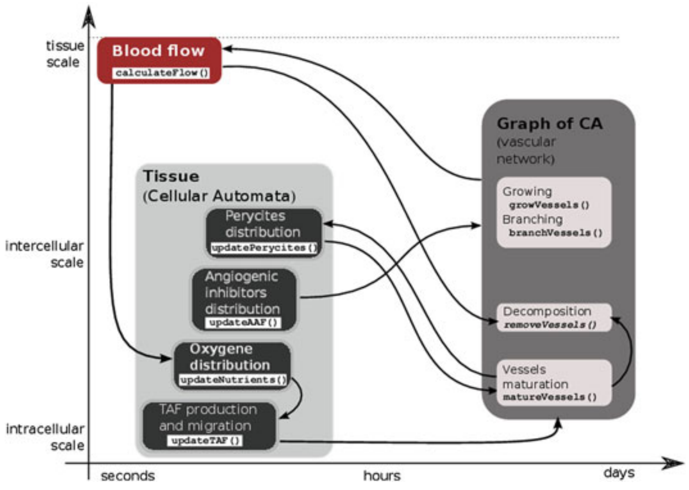


Fig. 1. Components of the model and dependencies between them. Figure also demonstrates the multiscale structure of the model

- G_{CA} — a planar, directed and acyclic graph defined as (V_G, E_G), where $V_G \subset Z^2$ and $E_G \subset Z^2 \times Z^2$ are finite sets of vertices and edges, respectively,
- $X_K(i,j)$ — Moore neighbourhood for the (i,j) cell in the regular mesh of automata,
- S — is the set of state vectors corresponding to each cell: $S = S_m \times S_g$,

S_m — states of CA cells	S_g — states of GCA cells
t_{ij} — indicates tumour cell	age_{ij} — maturation level
taf_{ij} — TAFs concentration	tip_{ij} — indicates "tip" cell
n_{ij} — nutrient and O_2 concent	$pres_{ij}$ — pressure value
per_{ij} — mural cells concent	$flow_{ij}$ — flow value
aaf_{ij} — AAF concentration	

Figure 2 illustrates how the vascular network is created by this model. Primary vessel represent predefined initial part of vascular network. TAFs distributed over the lattice activate cells that belong to the primary vessels and new sprouts are initiated. They grow attracted by TAFs gradient. The "tip" cell, located at the end of active sprout, governs its development. Active sprouts can also branch and create anastomosis.

The CA rules implement processes connected with distribution various substances in tissue surrounding vascular network:

- Each cell (normal and tumour) every time-step consumes a certain amount of oxygen/nutrients.
- Tumour cells requires a certain level of oxygen and nutrients. Otherwise they start to produce TAFs. TAFs are distributed in the neighbourhood.

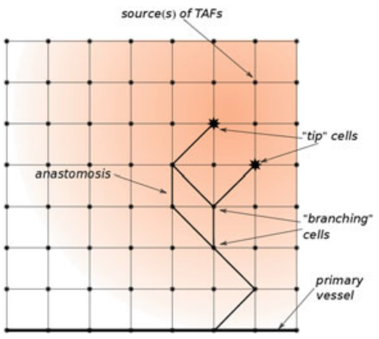


Fig. 2. Development of vascular network modelled with Graph of Cellular Automata

- Sources of AAFs are arbitrary defined, depending on a simulation scenario. They are also distributed to establish a certain gradient.
- Nutrients/oxygen are produced by cells that belong to graph, are mature and transport blood.
- It is assumed that pericytes are uniformly scattered in the tissue and during the maturation of blood are used.

The distribution of various substances (TAFs, AAFs, nutrient/oxygen etc.) are modelled with CA rule implementing Laplace equation.

The following rules are applied to drive the development of the vessel network:

1. Cells in existing vessel are activated by a certain concentration of TAFs.
2. Sprouts start at activated cells and grow attracted by TAFs gradient.
   - the growth of sprout is governed by "move" of their active end called "tip" — at each step of simulation, for each "tip" cell a successor cell is calculated.
   - the reinforced random walk model [11] is used to find a new "tip" cell.
3. Anastomosis is created when one sprout meet another.
4. Initially, a new cell included to the graph has state immature. Then, in the consecutive steps of simulation, maturity level increases until it reaches a mature state. The rate of maturation depends on local pericytes concentration as well as TAFs and AAFs concentration.
5. Vessels (or part of vessels) that cannot mature and transport blood are removed after a certain period of time.
6. AAFs prevent sprout growing/branching/maturing if a certain concentration threshold is exceeded.

3.1 Blood Flow Calculations in Modelled Vasculature

Blood flow is calculated in the separated module that takes the graph of vessels as input. Flow calculations are proceeded whenever any changes in the graph structure occurs. In each time-step after updating graph structure new loops are calculated by using the following algorithm:

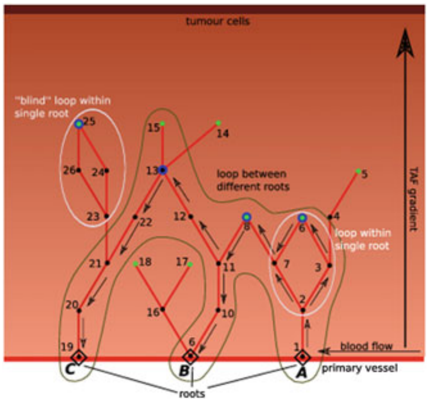


Fig. 3. Calculating closed loops in vascular network. Green dots represent "tip" cells. "Closing" nodes are marked in blue (Color figure online).

1. searching for closed loops in the graph,
2. setting and solving the system of equations for new pressures distribution in graph nodes,
3. calculating flows in the graph according to the pressure's distribution.

Initially, closed loops are detected in graph structure generated by the model (see Fig. 3). "Blind" loops that start and ends in the same node, are eliminated from the calculation.

Following the approach proposed by Mcdougall, Sherratt, Anderson i Chaplain [17], the Poiseuille law is applied for modeling flow in a single segment connecting nodes i and j.

$$Q_{ij} = \frac{\pi R_{ij}^4 \Delta P_{ij}}{8\mu L_{ij}} \tag{1}$$

- R_{ij} — segment diameter,
- L_{ij} — segment length,
- μ — viscosity,
- $\Delta P_{ij} = P_i - Pj$ — pressure difference between nodes i and j.

Arbitrary values of pressure for the "root" nodes are set at the beginning of simulation. These nodes do not change their pressure values during the simulation and force the flow. In order to calculate pressures in all nodes participating in flow the system of equations is constructed:

$$\begin{cases} a_{11}P_1 + a_{12}P_2 + ... + a_{1n}P_n = b_1 \\ a_{21}P_1 + a_{22}P_2 + ... + a_{2n}P_n = b_2 \\ \dots \\ a_{mn}P_1 + a_{m2}P_2 + ... + a_{mn}P_n = b_m \end{cases} \tag{2}$$

where the coefficients of the system are defined as follows:

$$a_{ij} = \begin{cases} 0 & \text{if } i \text{ and } j \text{ are not neighbouring nodes,} (i \neq j) \\ -\frac{R_{ij}^4}{L_{ij}} & \text{if } i \text{ i } j \text{ are neighbouring nodes,} \\ \sum_{k=1}^{k=s} \frac{R_{ik}^4}{L_{ik}} & \text{if } i = j \text{ and } s \text{ is a number of neighbours} \end{cases}$$

and the constant terms are:

$$b_i = \begin{cases} 0 & \text{if there is no roots in neighbourhood of node } i, \\ \sum_{k=1}^{k=s} \frac{R_{ik}^4}{L_{ik}} & \text{otherwise and } s \text{ is a number of root neighbours} \end{cases}$$

The solution of the system of equations is the pressure distribution. Flows in the segments of network are calculated by using Poiseuille equation (1).

3.2 The Implementation

Models based on CA paradigm are conceptually simple but require a large amount of data to represent a modelled system. In order to achieve satisfactory performance it is necessary to parallelize the code. The GCA model operates on variety of data structure regular (2D or 3D meshes) as well as irregular (graphs, linked trees, etc.). Thus, it is difficult to write parallel code that achieves high efficiency. The OpenMP library was chosen to implement the algorithms. As the processing of Cellular Automata is naturally parallel, parallelization is achieved automatically. Procedures that operate on Graph of Cellular Automata have been parallelized too. The network is partitioned into subtree and assigned to different threads.

The quality of parallelization has been tested on two machines with shared memory: 4 Intel Pentium processor and 8 AMD/Opterons processors. Figure 5 present speed-ups achieved on both the machines. As the significant parts of computation are concerned with irregular structure of the graph, the performance is far from almost linear speed-up typical for pure Cellular Automata computations.

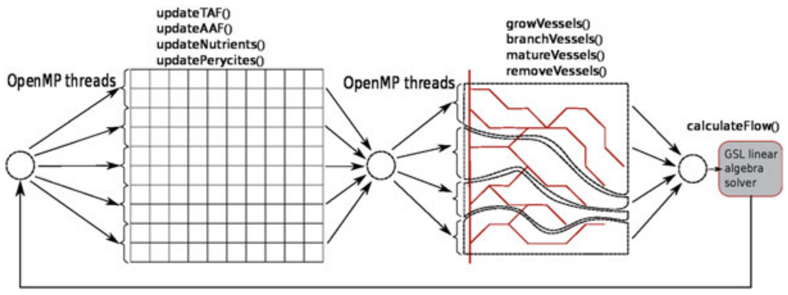


Fig. 4. Outline of the parallel implementation of the tumour angiogenesis model

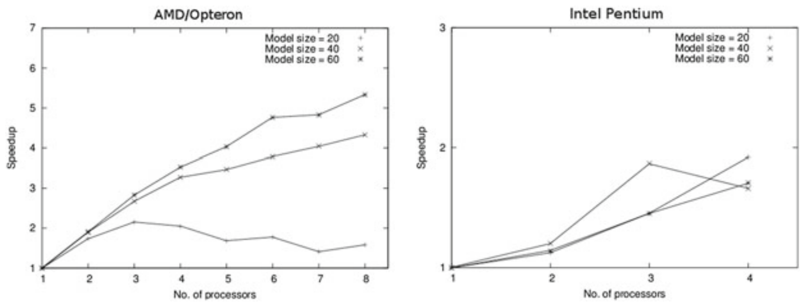


Fig. 5. Speed-ups measured on 4-processor and 8-processor Opteron machines for different model size.

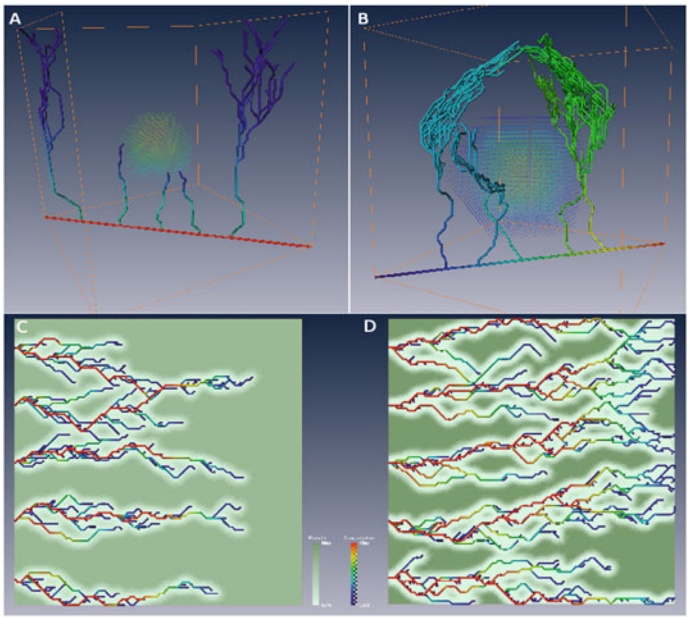


Fig. 6. Simulation results demonstrates influence of inhibitors (A, B) and maturation factors (C, D)

The algorithms that operate on CA lattice scale almost linearly. Overall efficiency is greatly reduced due to the very poor scalability of algorithms operating on the graph. The main factor is irregular schema of communication between computational nodes when the graph is updated.

4 Results

Our implementation is able to handle various configurations and simulation scenarios in two and three dimensions. Below we present sample results.

Figure 6 presents how defined rules of inhibitors influence the results. Inhibitors in configuration A prevent vessels from maturation. According to other defined rules, the capillaries that penetrate a region with inhibitors cannot mature and their development is completely inhibited. Figure 6B presents the case when inhibiting factor blocks the migration of endothelial cells. As the result, the network develops in that way that omits the region with inhibitors.

Figure 6C, D demonstrates the influence of pericytes on maturation of vascular network. The pericytes are initially uniformly distributed on the mesh. Newly formed capillaries collects pericytes which are used to cover and stabilize them. In Fig. 6A we observe simulation results for configuration with a lower initial amount of pericytes. The network is relatively small, sparse and mostly immature. Figure 6B presents how the increasing of the initial amount of pericytes influences the results (other parameters are the same as in B). The network grow faster and reach the other edge of mesh, where tumour cells are located.

5 Conclusions

This paper summarizes model of Tumour Induced Angiogenesis based on Graph of Cellular Automata paradigm. The unique feature of the model is the explicit use of a graph as a representation of the network of blood vessels. In this approach, a Cellular Automata model is extended with graph structure. Such the construction gives us the ability to apply the different computational approaches to these components in order to achieve highest efficiency.

The structure of the graph, which represents the vascular network also supports the modeling of blood flow. This component allows to more precisely predict how newly formed vascular network is able to supply nutrients/oxygen. Additionally, such the results can be useful for investigations on cancer therapies that are targeted on normalization of vasculature. The efficient network of vessels can also carry drugs that destroy cancer cells.

The application of graphs also facilitate quantitative verification of simulation results [18]. The vascular network encoded in graphs can be easily described by using various network descriptors.

Acknowledgements. This research is partially supported by AGH-University of Science and Technology grant No. 11.11.230.015.

References

1. Carmeliet, P.: Angiogenesis in life, disease and medicine. Nature **438**, 932–936 (2005)
2. Mantzaris, N., Webb, S., Othmer, H.G.: Mathematical modeling of tumour-induced angiogenesis. J. Math. Biol. **49**(2), 111–187 (2004)
3. Rew, D.A.: Modelling in tumour biology part 1: modelling concepts and structures. Eur. J. Surg. Oncol. **26**(1), 87–94 (2000)

4. Bergers, G., Song, S.: The role of pericytes in blood-vessel formation and maintenance. Neuro-oncology **7**(4), 452–464 (2005)
5. Topa, P., Dzwinel, W., Yuen, D.: A multiscale cellular automata model for simulating complex transport systems. Int. J. Mod. Phys. C **17**(10), 1–23 (2006)
6. Topa, P.: Towards a two-scale cellular automata model of tumour-induced angiogenesis. In: El Yacoubi, S., Chopard, B., Bandini, S. (eds.) ACRI 2006. LNCS, vol. 4173, pp. 337–346. Springer, Heidelberg (2006)
7. Topa, P.: Dynamically reorganising vascular networks modelled using cellular automata approach. In: Umeo, H., Morishita, S., Nishinari, K., Komatsuzaki, T., Bandini, S. (eds.) ACRI 2008. LNCS, vol. 5191, pp. 494–499. Springer, Heidelberg (2008)
8. Byrne, H., Chaplain, M.A.J.: Mathematical models for tumour angiogenesis: numerical simulations and nonlinear wave solutions. Bull. Math. Biol. **57**, 461–486 (1995)
9. Anderson, A.R., Chaplain, M.A.: A mathematical model for capillary network formation in the absence of endothelial cell proliferation. Appl. Math. Lett. **11**(3), 109–114 (1998)
10. Plank, M.J., Sleeman, B.D., Jones, P.F.: A mathematical model of tumour growth, regulated by vascular endothelial growth factor and the angiopoietins. J. Theor. Biol. **229**, 435–454 (2004)
11. Anderson, A.R., Chaplain, M.A.: Continuous and discrete mathematical models of tumour-induced angiogenesis. Bull. Math. Biol. **60**, 857–900 (1998)
12. Lowengrub, J.S., et al.: Nonlinear modelling of cancer bridging the gap between cells and tumours. Nonlinearity **23**(1), R1–R96 (2010)
13. Macklin, P., et al.: Multiscale modelling and nonlinear simulation of vascular tumour growth. J. Math. Biol. **58**, 765–798 (2009)
14. Stokes, C.L., Lauffenburger, D.A.: Analysis of the roles of microvessel endothelial cell random motility and chemotaxis in angiogenesis. J. Theor. Biol. **152**, 377–403 (1991)
15. Anderson, A.R.A., Pitcairn, A.: Application of the hybrid discrete-continuum technique. In: Alt, W., et al. (eds.) Polymer and Cell Dynamics-Multiscale Modeling and Numerical Simulations, pp. 261–279. Birkhauser, Basel (2003)
16. Owen, M.R., Alarcón, T., Byrne, H.M., Maini, P.K.: Angiogenesis and vascular remodelling in normal and cancerous tissues. J. Math. Biol. **58**(4), 689–721 (2009)
17. McDougall, S.R., et al.: Mathematical modelling of flow through vascular networks: implications for tumour-induced angiogenesis and chemotherapy strategies. Bull. Math. Biol. **64**(4), 673–702 (2002)
18. Topa, P., Dzwinel, W.: Using network descriptors for comparison of vascular systems created by tumour-induced angiogenesis. Theor. Appl. Inf. **21**(2), 83–94 (2009)

Neighborhood Selection and Rules Identification for Cellular Automata: A Rough Sets Approach

Bartłomiej Płaczek(✉)

Institute of Computer Science, University of Silesia, Będzińska 39,
41-200 Sosnowiec, Poland
Placzek.Bartlomiej@gmail.com

Abstract. In this paper a method is proposed which uses data mining techniques based on rough sets theory to select neighborhood and determine update rule for cellular automata (CA). According to the proposed approach, neighborhood is detected by reducts calculations and a rule-learning algorithm is applied to induce a set of decision rules that define the evolution of CA. Experiments were performed with use of synthetic as well as real-world data sets. The results show that the introduced method allows identification of both deterministic and probabilistic CA-based models of real-world phenomena.

Keywords: Rough sets · Cellular automata · Model identification

1 Introduction

Cellular automata (CA) have found many applications in the field of complex systems modeling. There is a number of works devoted to CA models for simulation of real-world phenomena like pedestrian dynamics [16], traffic flow [10], urban growth [6], etc. In most cases, the update rule and neighborhood for CA are determined by human experts that have knowledge of the modeled phenomenon. Automatic identification of CA-based models remains an open research issue.

Several attempts have been made in the literature to develop algorithms for CA identification. However, most of the previous research did not investigate the use of real-world data sets as an input. The available algorithms were designed and tested mainly against synthetic data obtained from CA evolution. The use of real-world data for identification of CA models poses additional challenges due to inherent complexity of modeled phenomena and errors that are made during data acquisition.

This paper discusses the possibility of using data mining techniques based on the rough sets theory [2] to select neighborhood and determine rules for CA models. According to the proposed approach, input data describing observed states of cells are represented in form of a decision table. Neighborhood is detected by using algorithms for reducts calculation. Cells that do not belong to the neighborhood are removed from the decision table. After that, a rule-learning method is applied to induce a set of decision rules that define the evolution of CA. This approach was tested in identification of both deterministic and probabilistic CA models for synthetic as well as real-world data sets.

2 Related Works

Most of the works related to the CA identification problem use genetic algorithms as a tool to extract update rule and neighborhood from spatio-temporal patterns produced in CA evolution [3,8,17]. In [12] a genetic algorithm was employed to learn probabilistic rules directly from experimental data for two-dimensional binary-state CA. Several methods were proposed that use genetic algorithms to learn CA rules and neighborhood size for image processing tasks [4,5].

Although application of genetic algorithms was a dominant approach to CA identification, some non-genetic techniques are also available. Adamatzky [1] proposed several approaches to extracting rules for different classes of CA. Straatman et al. [14] developed a form of hill climbing and backtracking to identify CA rules. In [13] a deterministic sequential floating forward search method was used to select rules of CA. Another approach is based on parameter estimation methods from the field of system identification [18]. A framework for solving the identification problems for both deterministic and probabilistic CA was presented in [15].

Main drawbacks of the existing CA identification methods are related to the fact that they were either designed for a particular class of CA, or their experimental evaluation was limited to the case of synthetic data. The introductory research results presented in this paper shows that the rough sets approach may be effectively used to develop a universal method for identification of CA models that mimic real-world phenomena.

3 Basic Concepts

Formally, a cellular automaton can be defined as a triple (V, N, δ), where V is a non-empty set of cell states, N is the neighbourhood, and δ is the update rule. Arguments of δ are the current states of cells in the neighbourhood, while the value of δ indicates the state of a central cell at the next time step.

The problem of CA identification involves finding both the cells neighborhood and the update rule on the basis of a training data set, which includes observed states of the cells. In order to use the rough sets approach for solving this problem, the training set of data has to be represented in the form of a decision table $I = (U, C)$, where U is a non-empty set of observations and C is a set of cell states: $C = \{c_\alpha(t), \ldots, c_i(t), \ldots, c_\omega(t), c_i(t+1)\}$.

State of j-th cell at time step t is denoted by $c_j(t)$. Index i indicates the central cell, for which neighborhood and update rule have to be found. Thus, the cell state $c_i(t+1)$ is used as the decision attribute. The remaining cell states $c_\alpha(t), \ldots, c_\omega(t)$ are condition attributes. The candidate neighbors $\alpha, \ldots, \omega$ are the cells for which distance to the central cell i is lower than a threshold value. The threshold is determined experimentally. Above definition can be easily extended to multidimensional CA by adding the necessary indexes.

Neighborhood for the i-th cell can be selected by calculating reducts of the above-defined decision table. A reduct is a subset of the condition attributes, which is sufficient to determine the decision attributes [2]. Taking into account the decision table discussed above, reduct should be defined as a subset of cell states $R \subseteq c_\alpha(t), \ldots, c_\omega(t)$, which preserves discernibility of the observations with regard to the decision $c_i(t+1)$, and none of its proper subsets has this ability. Observations are discernible if they differ in at least one condition attribute (cell state). Each two observations that have different decisions $c_i(t+1)$ and are discernible by the full set of cell states $c_\alpha(t), \ldots, c_\omega(t)$ are also discernible by the reduct R.

The neighborhood N of a cell is determined as a set of cells, whose states belongs to the reduct $R : N = \{j : c_j(t) \in R\}$, where j is a cell index. There may exist multiple reducts for one decision table. Selection of the neighborhood is made with regard to the shortest reduct, because size of the neighborhood has to be minimized. If there are several minimal reducts then the one is selected which has the lowest average distance between neighbors and the central cell. When the reduct R is found, the condition attributes that do not belong to this reduct are excluded from the decision table. Thus, the modified decision table $I' = (U, C')$ has the following set of attributes: $C' = R \cup \{c_i(t+1)\}$, where $c_i(t+1)$ remains the decision attribute.

Update rule of a cellular automaton is identified as a set of decision rules by taking into account the information from the modified decision table I'. A particular decision rule r has the following form:

$$(c_x(t) = v_x) \wedge \ldots \wedge (c_y(t) = v_y) \Rightarrow c_i(t+1) = v_i \tag{1}$$

where: $\{c_x(t), \ldots, c_y(t)\} \subseteq R$, $v_j \in V$, and V denotes the set of allowable cell states.

Two characteristics of decision rules are useful for the proposed method: support and match [2]. Support of rule r, denoted by $SUPP_{I'}(r)$, is equal to the number of observations from I' for which rule r applies correctly, i.e., premise of the rule is satisfied and the decision given by rule is consistent with the one in decision table. $MATCH_{I'}(r)$ is the number of observations in I' for which the rule r applies, i.e., premise of the rule is satisfied. Based on these two characteristics, a certainty factor is defined for the decision rule r:

$$CER_{I'}(r) = SUPP_{I'}(r)/MATCH_{I'}(r). \tag{2}$$

The certainty factor may be interpreted as a conditional probability that decision of rule r is consistent with an observation in the decision table I', given that premise of the rule is satisfied.

Figure 1 summarizes the main operations that are necessary to identify a cellular automaton by using the rough sets approach. In this study, reducts and decision rules are calculated using algorithms implemented in the RSES software [2]. Three algorithms of reducts calculation were examined: exhaustive, genetic,

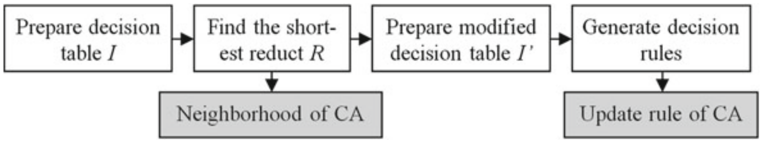

Prepare decision table I
Find the shortest reduct R
Prepare modified decision table I'
Generate decision rules
Neighborhood of CA
Update rule of CA

Fig. 1. Rough sets based procedure of CA identification

and dynamic reduct algorithm. Moreover, the experiments involved application of three algorithms that enable induction of decision rules: exhaustive, genetic, and LEM2.

4 Identification of Deterministic Cellular Automata

In this section, the proposed approach is applied for identification of three deterministic CA [9]: elementary cellular automaton with Wolfram's rule 184 (ECA-184), deterministic version of Nagel-Schreckenberg cellular automaton (NaSch-D), and the Conway's game of life cellular automaton (Life).

ECA-184 is a one-dimensional cellular automaton with binary cell states and neighborhood of three cells wide. Original definition of the ECA-184 update rule is presented in Fig. 2. The upper row in this figure illustrates all possible states of a central cell and its neighborhood. Lower row shows states of the central cell after update – in the next time step of the CA evolution.

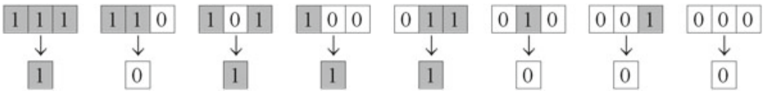


Fig. 2. Update rule of ECA-184

Identification of ECA-184 was performed using a training data set of 500 observations. Each observation in the decision table I has covered a group of 21 cells (candidate neighbors) and the 11-th cell was considered as the central one. The neighborhood of ECA-184 was correctly recognized by each of the reducts calculation algorithms (exhaustive, genetic, and dynamic). The shortest reduct was determined as $R = \{c_{i-1}(t), c_i(t), c_{i+1}(t)\}$, thus the modified decision table I' had four attributes: $C' = \{c_{i-1}(t), c_i(t), c_{i+1}(t), c_i(t+1)\}$. Table 1 presents the decision rules that were calculated from table I' by using the LEM2 algorithm. Symbol $\emptyset$ indicates that a given cell state does not occur in a particular decision rule, e.g., the rule no. 3 from Table 1 should be interpreted as: $(c_{i-1}(t) = 1) \wedge (c_i(t) = 0) \Rightarrow c_i(t+1) = 1$. The set of decision rules in Table 1 is consistent with the original update rule of ECA-184 (Fig. 2). The remaining algorithms of rule induction (exhaustive and genetic) have also generated valid sets of decision rules; however their size was larger (6 rules).

Table 1. Decision rules generated for ECA-184

Rule no.	1	2	3	4	5
$c_{i-1}(t)$	0	0	1	$\emptyset$	1
$c_i(t)$	$\emptyset$	0	0	1	1
$c_{i+1}(t)$	0	$\emptyset$	$\emptyset$	1	0
$c_i(t+1)$	0	0	1	1	0

The second CA identification example concerns a deterministic version of the Nagel-Schreckenberg model for road traffic simulation (NaSch-D) [9]. Update rule of NaSch-D consists of two steps: (I) acceleration and braking of vehicle, (II) vehicle movement. In step I velocity $v_k(t)$ for each vehicle (k) is calculated in cells per time step: $v_k(t) \leftarrow \min\{v_k(t-1)+1, g_k(t-1), v_{\max}\}$, where g_k is the number of empty cells in front of vehicle k, $v_{\max}$ denotes maximal velocity of vehicles. Step II simulates movement of the vehicles – index of a cell occupied by vehicle k at time step t, denoted by $x_k(t)$, is determined using the formula $x_k(t) \leftarrow x_k(t-1) + v_k(t)$.

In this study, the parameter v_{max} was set to 2 cells per time step. Thus, the set of cell states includes 4 values: $c_i(t) = -1$ denotes empty cell, and $c_i(t) = 0, \ldots, 2$ indicates velocity of the vehicle that occupies the i-th cell. The training data set was prepared in the same way as for the previous example. Decision table I contained 2000 observations. The shortest reduct $R = \{c_{i-2}(t), \ldots, c_{i+2}(t)\}$ was uniquely determined by all the examined algorithms. Table 2 shows the set of 16 decision rules generated for NaSch-D by using LEM2 algorithm. In case of exhaustive as well as genetic algorithm the number of obtained decision rules was 37.

Table 2. Decision rules generated for NaSch-D

Rule no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$c_{i-2}(t)$	−1	−1	$\emptyset$	$\emptyset$	0	$\emptyset$	2	1	$\emptyset$	2	$\emptyset$	2	2	2	2	2
$c_{i-1}(t)$	$\emptyset$	−1	$\emptyset$	$\emptyset$	$\emptyset$	1	−1	−1	1	−1	1	−1	−1	−1	−1	−1
$c_i(t)$	$\emptyset$	−1	0	0	$\emptyset$	−1	−1	$\emptyset$	−1	−1	−1	−1	−1	−1	−1	−1
$c_{i+1}(t)$	−1	$\emptyset$	0	$\emptyset$	−1	0	−1	−1	2	0	1	2	−1	−1	1	−1
$c_{i+2}(t)$	$\emptyset$	$\emptyset$	$\emptyset$	−1	$\emptyset$	$\emptyset$	−1	$\emptyset$	−1	$\emptyset$	−1	$\emptyset$	2	0	$\emptyset$	1
$c_i(t+1)$	−1	−1	0	1	−1	0	2	−1	2	0	1	2	2	1	1	2

The set of decision rules in Table 2 has the smallest size. However, its application requires additional assumptions to resolve conflicts when various rules that match the observed state of the neighborhood suggest different states of the central cell. This issue was addressed by using priorities of the decision rules. In case of conflict, the rule with the highest priority is selected. The priority is determined taking into account support of the rule. A rule with a larger support has higher priority. In Table 2, the decision rules are sorted in descending order, according to their priorities. It was verified that for each possible state of the neighborhood, the above defined decision algorithm enables correct update of the central cell.

Next, the proposed method was applied to identify the two dimensional CA called Life [9]. In this example, the binary cell states $c_{i,j}(t) = 0$ or $c_{i,j}(t) = 0$ indicates that the cell (i, j) is dead or alive respectively. Every cell interacts with its eight neighbors (Moore neighborhood). At each time step, the following updates occur: (1) Any live cell with fewer than two live neighbors dies. (2) Any live cell with two or three live neighbors lives on to the next generation. (3) Any live cell with more than three live neighbors dies. (4) Any dead cell with exactly three live neighbors becomes a live cell.

Observations collected in a decision table for identification of the Life CA have included states of 25 cells from the Moore neighborhood of radius 2. The actual neighborhood that includes 9 cells was correctly recognized by the reducts calculation algorithms. However, taking into account the states of all 9 cells from the neighborhood, a large set of decision rules was obtained (154–242 rules, depending on the algorithm). In order to decrease the number of decision rules, the amount of live neighbors L was added as an attribute of the decision table. After this modification, the reducts calculation algorithms were executed again, and the resulting shortest reduct has included only two attributes: $R = \{c_{i,j}(t), L\}$. Thus, a decision table I' with attributes $C' = \{c_{i,j}(t), L, c_{i,j}(t+1)\}$ was used to generate decision rules. The set of rules in Table 3 was generated by the exhaustive rule induction algorithm. These decision rules are consistent with the update rule of Life, which was described above. In case of LEM2 algorithm, the rule set had 17 elements. The genetic algorithm has provided an incomplete set of decision rules that fail to describe the evolution of Life.

Table 3. Decision rules generated for Life

Rule no.	1	2	3	4	5	6	7	8	9	10
$c_{i,j}(t)$	$\emptyset$	$\emptyset$	0	1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
L	0	1	2	2	3	4	5	6	7	8
$c_{i,j}(t+1)$	0	0	0	1	1	0	0	0	0	0

5 Identification of Probabilistic Cellular Automata

In case of probabilistic CA, the identification task is more complicated because the state of neighborhood does not uniquely determine the state of central cell. Therefore, some of the decision rules that describe evolution of a probabilistic CA are uncertain. It means that there exist decision rules that have identical premises and give different decisions. According to the proposed approach, such rules are merged into one rule whose decision is defined using a set of pairs of vales v_z and certainty factors $CER_{I'}(r_z)$:

$$(c_x(t) = v_x) \wedge \ldots \wedge (c_y(t) = v_y) \Rightarrow c_i(t+1) = \{v_z/CER_{I'}(r_z)\}, \qquad (3)$$

where r_z indicates the decision rule $(c_x(t) = v_x) \wedge \ldots \wedge (c_y(t) = v_y) \Rightarrow c_i(t+1) = v_z$.

During update operation of a CA, if the premise $(c_x(t) = v_x) \wedge \ldots \wedge (c_y(t) = v_y)$ is satisfied then the state of a central cell $c_i(t+1)$ takes value v_z with probability $CER_{I'}(r_z)$. This method utilizes the fact that certainty factor $CER_{I'}(r_z)$ may be interpreted as a conditional probability (see Sect. 3). It should be also noted that for the rule defined by (3) the condition $\Sigma_z CER_{I'}(r_z) = 1$ is always satisfied.

The rough sets approach was applied to identification of the Nagel-Schreckenberg probabilistic cellular automaton (NaSch) [9]. Deterministic version of NaSch was described in Sect. 4. For probabilistic NaSch, the first step of update rule is extended by step I-a, which includes randomization of the velocity according to formula $\xi(t) < p \Rightarrow v_k(t) \leftarrow \max\{0, v_k(t) - 1\}$, where: $\xi \in [0, 1)$ is a random number drawn from a uniform distribution, and $p \in [0, 1]$ is a parameter called braking probability.

Traffic simulation was performed by using the NaSch model with parameters $v_{max} = 1$ and $p = 0.2$. The binary cell states were used in this example to distinguish empty and occupied cells. A decision table was prepared based on observations that were collected during the simulation. Reducts of the decision table were calculated to select the neighborhood. Due to the existence of conflicting observations, the genetic algorithm did not find any reducts and reducts calculated by both the exhaustive and the dynamic algorithm have included a number of cell states that in fact are not taken into account by the update rule of the analyzed CA. It was necessary to use a reduct shortening algorithm [2] for correct determination of the neighborhood. After shortening operation, the reduct $R = \{c_{i-1}(t), c_i(t), c_{i+1}(t)\}$ was found, which is consistent with the actual neighborhood of cells in NaSch. The set of decision rules generated by LEM2 algorithm (Table 4) is equivalent to the update rule of NaSch. For remaining algorithms incomplete sets of the decision rules were obtained.

Table 4. Decision rules generated for NaSch ($v_{max} = 1$)

Rule no.	1	2	3	4	5	6	7
$c_{i-1}(t)$	0	0	0	1	1	1	1
$c_i(t)$	0	1	1	0	0	1	1
$c_{i+1}(t)$	$\emptyset$	0	1	0	1	0	1
$c_i(t+1)$	0/1	0/0.8, 1/0.2	1/1	0/0.2, 1/0.8	0/0.2, 1/0.8	0/0.8, 1/0.2	1/1

The proposed approach was also verified in identification of NaSch model with parameter $v_{max} = 2$, however the results are not presented here due to space limitations.

6 Identification of Cellular Automata Using Real-World Data

In this section the rough sets approach is used to identify a CA on the basis of real-world observations regarding shock waves that emerge in highway traffic. The analyzed traffic data, describing vehicles trajectories in time space diagrams, were taken from [19]. Training data set includes cell states that correspond to the positions of vehicles determined from the time space diagrams. It was assumed that one cell represents a 7 m segment of traffic lane. State of a cell is defined as a binary value: 0 denotes empty cell and 1 refers to an occupied cell. An example of the training data is presented in Fig. 3(a).

Neighborhood and update rule were determined using the method devised for probabilistic CA in previous section. The algorithm for finding dynamic reducts was applied and followed by the reduct shortening operation. The resulting reduct includes states of four cells: $R = \{c_{i-2}(t), c_{i-1}(t), c_i(t), c_{i+1}(t)\}$. Decision rules were generated by using the LEM2 algorithm. The obtained rules (Table 5) do not guarantee that the number of occupied cells (vehicles) will be constant during the CA evolution. This fact leads to unrealistic traffic simulation.

Table 5. Decision rules generated from real-world traffic data

Rule no.	1	2	3	4	5	6	7	8
$c_{i-2}(t)$	0	0	0	0	0	0	0	0
$c_{i-1}(t)$	0	0	0	0	1	1	1	1
$c_i(t)$	0	0	1	1	0	0	1	1
$c_{i+1}(t)$	0	1	0	1	0	1	0	1
$c_i(t+1)$	0/1	0/1	0/1	1/1	0/0.35, 1/0.65	0/0.2, 1/0.8	0/0.5, 1/0.5	1/1
Rule no.	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
$c_{i-2}(t)$	1	1	1	1	1	1	1	1
$c_{i-1}(t)$	0	0	0	0	1	1	1	1
$c_i(t)$	0	0	1	1	0	0	1	1
$c_{i+1}(t)$	0	1	0	1	0	1	0	1
$c_i(t+1)$	0/0.75, 1/0.25	0/0.75, 1/0.25	0/0.8, 1/0.2	1/1	0/0.5, 1/0.5	0/0.5, 1/0.5	0/0.3, 1/0.7	1/1

The unrealistic model behavior may occur because the decisions for uncertain rules are made randomly, without any coordination. E.g., the rules 7 and 13 in Table 5 apply to the same configuration of cells $(0, 1, 1, 0, 0)$. If for both rules the random decision is 1 then the resulting configuration $(0, 1, 1, 1, 0)$ includes three instead of two occupied cells. This problem was resolved by adding error detection rules to the CA update algorithm. If error is detected by rules (4) then the update of cells $(i-2, i-1, i)$ is repeated.

$$\begin{aligned}
&(c_{i-2}(t)=1)\wedge(c_{i-1}(t)=0)\wedge(c_i(t)=0)\wedge\\
&\wedge(c_{i-2}(t+1)+c_{i-1}(t+1)+c_i(t+1)\neq 1)\Rightarrow \text{error}=\text{true}\\
&(c_{i-2}(t)=1)\wedge(c_{i-1}(t)=0)\wedge(c_i(t)=1)\wedge\\
&\wedge(c_{i-2}(t+1)+c_{i-1}(t+1)\neq 1)\Rightarrow \text{error}=\text{true}
\end{aligned}\tag{4}$$

Above modifications allow the CA to perform realistic simulation of traffic flow. Figure 3 presents time space diagrams for two shock waves. The shock wave (a) was observed in real highway traffic and the shock wave (b) was obtained from evolution of the identified CA. This example illustrates the possibility of using the rough sets approach for identification of CA-based models of real-world phenomena.

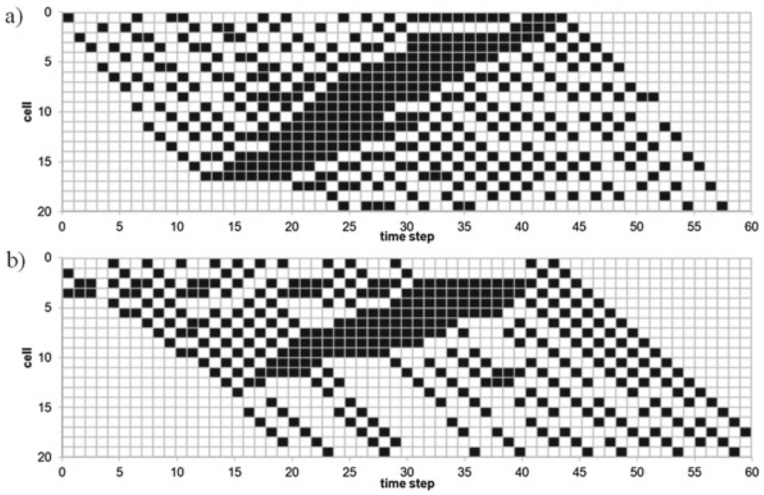


Fig. 3. Shock waves: (a) real-world data, (b) results of CA evolution

7 Conclusions

Results of the introductory research show that the data exploration techniques based on rough sets theory enable proper selection of neighborhood and update rule induction for different classes of CA. It was also demonstrated that the rough sets approach is suitable for CA models identification from real-world data sets. The best results were obtained for the proposed CA identification procedure when implemented by using the dynamic reducts algorithm for neighborhood selection and the LEM2 algorithm for decision rules induction. Nevertheless, in case of stochastic CA identification, some additional actions were necessary (reduct shortening, adding error detection rules). Verification of the usability of the proposed approach for a wider set of data as well as comparison with state-of-art methods based on genetic algorithms remain open issues for further research. Another interesting topic for future studies is to apply the proposed method in a data exploration system for CA-based image processing [7,11].

References

1. Adamatzky, A.: Identification of Cellular Automata. T&F, London (1994)
2. Bazan, J., Szczuka, M.S.: The rough set exploration system. In: Peters, J.F., Skowron, A. (eds.) Transactions on Rough Sets III. LNCS, vol. 3400, pp. 37–56. Springer, Heidelberg (2005)
3. Billings, S., Yang, Y.: Identification of probabilistic cellular automata. IEEE Trans. Syst. Man Cybern. B Cybern. **33**(2), 225–236 (2003)
4. Chavoya, A., Duthen, Y.: Using a genetic algorithm to evolve cellular automata for 2D/3D computational development. In: Genetic and Evolutionary Computation Conference, pp. 231–232 (2006)
5. Craiu, R.V., Lee, T.C.M.: Pattern generation using likelihood inference for cellular automata. IEEE Trans. Image Process. **15**(7), 1718–1727 (2006)
6. Feng, Y., Liu, Y., Tong, X., Liu, M., Deng, S.: Modeling dynamic urban growth using cellular automata and particle swarm optimization rules. Landscape Urban Plan. **102**(3), 188–196 (2011)
7. Ładniak, M., Piórkowski, A., Młynarczuk, M.: The data exploration system for image processing based on server-side operations. In: Saeed, K., Chaki, R., Cortesi, A., Wierzchoń, S. (eds.) CISIM 2013. LNCS, vol. 8104, pp. 168–176. Springer, Heidelberg (2013)
8. Maeda, K.-I., Sakama, C.: Identifying cellular automata rules. J. Cell. Autom. **2**(1), 1–20 (2007)
9. Maerivoet, S., De Moor, B.: Cellular automata models of road traffic. Phys. Rep. **419**, 1–64 (2005)
10. Płaczek, B.: Fuzzy cellular model for on-line traffic simulation. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009, Part II. LNCS, vol. 6068, pp. 553–560. Springer, Heidelberg (2010)
11. Płaczek, B.: Rough sets in identification of cellular automata for medical image processing. J. Med. Inf. Technol. **22**, 161–168 (2013)
12. Richards, F.C., Meyer, T.P., Packard, N.H.: Extracting cellular automaton rules directly from experimental data. Phys. D **45**(1–3), 189–202 (1990)
13. Rosin, P.: Training cellular automata for image processing. IEEE Trans. Image Process. **15**(7), 2076–2087 (2006)
14. Straatman, B., White, R., Engelen, G.: Towards an automatic calibration procedure for constrained cellular automata. Comput. Environ. Urban Syst. **28**(1–2), 149–170 (2004)
15. Sun, X., Rosin, P.L., Martin, R.R.: Fast rule identification and neighborhood selection for cellular automata. IEEE Trans. Syst. Man Cybern. B Cybern. **41**(3), 749–760 (2011)
16. Was, J.: Cellular automata model of pedestrian dynamics for normal and evacuation conditions. In: 5th International Conference on Intelligent Systems Design and Applications, ISDA'05, pp. 154–159. IEEE Press (2005)
17. Yang, Y., Billings, S.: Extracting Boolean rules from CA patterns. IEEE Trans. Syst. Man Cybern. B Cybern. **30**(4), 573–580 (2000)
18. Zhao, Y., Billings, S.: The identification of cellular automata. J. Cell. Autom. **2**(1), 47–65 (2007)
19. Coifman, B.: Time space diagrams for thirteen shock waves. Working Papers, California Partners for Advanced Transit and Highways (1997)

Coupling Lattice Boltzmann Gas and Level Set Method for Simulating Free Surface Flow in GPU/CUDA Environment

Tomir Kryza(✉) and Witold Dzwinel

Department of Computer Science, AGH University of Science and Technology, Kraków, Poland
{tomir,dzwinel}@agh.edu.pl

Abstract. We present here a proof-of-concept of a novel, efficient method for modeling of liquid/gas interface dynamics. Our approach consists in coupling the lattice Boltzmann gas (LBG) and the level set (LS) methods. The inherent parallel character of LBG accelerated by level sets is the principal advantage of our approach over similar particle based solvers. Consequently, this property allows for efficient use of our solver in GPU/CUDA environment. We demonstrate preliminary results and GPU/CPU speedups simulating two standard free surface fluid scenarios: the falling droplet and the breaking dam problems.

Keywords: Free surface flow · Lattice Boltzmann gas · Level sets · CUDA · GPGPU

1 Introduction

Computational fluid dynamics (CFD) often involves solving free surface problems such as river flows, floods and breaking waves. Important industrial problems include processes such as foaming and casting, inkjet droplet formation and various types of fluid-solid structure interactions. These free surface scenarios become always tough problems in terms of computational complexity. Thus, real-time simulation of two-phase (liquid/gas) flows is still a challenging goal. Meanwhile, the possibility of interactive visualization and simulation of approximate dynamics of the liquid/gas interface is in scope of great interest of game and simulator designers. Using GPU boards and CUDA technology for modeling free surface dynamics is the straightforward way for both speeding up the computations and increasing their precision.

When constructing approximate models of free surface fluid flow, the following basic aspects should be taken into account:

- dynamics of the liquid volume, i.e., calculation of velocity and pressure fields inside the liquid volume;
- interaction of liquid with container walls;
- representation of the free surface interface;

- dynamics of the liquid on the interface;
- liquid volume separation and merging, e.g., a droplet pinch-off or coalescence.

For the sake of efficiency, our model does not take into account internal dynamics of the gaseous phase (i.e., we handle the free surface conditions only at the interface).

The dynamics of fluid flow can be simulated classically by solving the Navier-Stokes equation numerically or by means of discrete particles methods. However, additional constrains that have to be imposed make these approaches computationally very demanding for simulating free surface flows. The CFD problems can also be attacked by solving discrete Boltzmann equation by using, so called, lattice Boltzmann gas (LBG) method [15]. In many cases, especially, for simulating complex fluids, LBG can be competitive in terms of efficiency for classical methods [2,18] mainly due to its inherent parallelism. On the other hand, the LBG approach fails to resolve thin layers of liquid efficiently [16]. Nevertheless, the Level Set Method [12] is a perfect modeling tool for tracking interface dynamics in the course of a simulation. It enables realistic and smooth representation of the liquid/gas interface diminishing computational load.

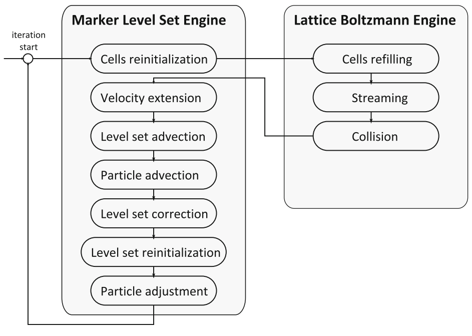
The main purpose of this paper is to present a novel concept which consists in coupling of LBG and level set methods. We expect that such a coupling can increase the overall efficiency of the free surface simulation. In Sect. 2 we show that this hybrid approach can be efficiently implemented on GPU architecture. In Sect. 3 we demonstrate preliminary results of modeling and GPU/CPU speedups employing two standard free surface fluid tests: the falling droplet and the breaking dam problems. We discuss the advantages of our approach in the Conclusions.

2 Algorithms and Implementation

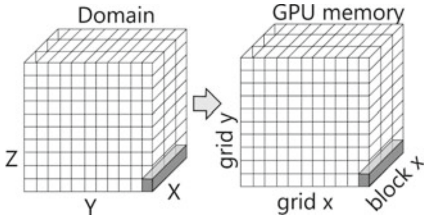
The coupling of the lattice Boltzmann gas and Marker Level Set [11] methods and implementation of this integrated approach in GPU/CUDA environment for approximate and fast simulation of the free surface fluid dynamics, is the main idea of this paper.

The process of modeling consists of steps that are depicted in Fig. 1(a). We model the free surface flow of incompressible fluid in a rectangular container. We assume that the friction between the fluid and the container walls is modeled with no-slip boundary conditions. For simplification reasons and to reduce the computational load we assume, without loss of generality, a fixed cell size $\Delta x = 1$ for LBG lattice and time step $\Delta t = 1$ for LBG and LS advection steps.

As shown in Fig. 1(a), the Marker Level Set mode consists of several steps processing the level set distance function, cell types matrix and marker particles set. The GPU kernels used for these computations can be divided into two groups: particle kernels and level set kernels. The level set kernels operate on a geometry of discrete lattice nodes while the particle kernels perform computations on the array of particle positions. The geometry of the level set kernels is structured as follows.



(a)



(b)

Fig. 1. (a) The block diagram of LBG and LS coupling strategy; (b) Mapping of the domain cells to GPU memory layout for LBG and LS kernels.

Each CUDA block contains a complete row of cells in x-axis. Domain dimensions y and z are mapped to the CUDA grid x and y indices respectively Fig. 1(b). Such layout enables efficient value acquiring from neighboring cells thanks to the L1/L2 cache on the Fermi architecture while keeping kernel implementation simple and easy to maintain.

The particle kernels layout is based on a mapping of grid/block hierarchy on one-dimensional array of structures. Each block contains a fixed number of threads ($K = 512$) in one row of the block. The grid contains one row of blocks. Number of blocks is defined by total number of particles: $blockCount = \frac{particleCount}{512}$. It is assumed that the number of particles is a multiple of the block size K. Separation of particles into blocks is dictated by the CUDA constraint on a maximum block size. The algorithms implemented in particle kernels do not use information from other particle than the processed one.

As shown in Fig. 1(a), the LBG mode is composed of steps responsible directly for simulation of liquid dynamics. All of GPU kernels used by this modeling mode operate on a similar setup as the level set kernels. Each block contains one full row of cells in x-axis while a plane of blocks maps onto y-z plane. This layout was inspired by implementation of a 3D lattice Boltzmann gas method implemented on a GPU described in [17] and was chosen because it yields a high memory throughput of LBG lattice processing. The components of the two computational modes are discussed below.

Cells Reinitialization. The zero level set isosurface defines the liquid/gas interface at the beginning of simulation. The following timesteps are based on the assumption that none of the *liquid cells* have a neighboring *gas cell*. Therefore, a reinitialization step is required that will define *interface cell* type. Such separation setup is presented in Fig. 2(a). The algorithm for defining this layer is as follows:

1. Initialize all boundary cells with **B**. For all other cells:
2. if $\Phi > 0$ assign cell type: **A**,

3. if $\Phi \leq 0$ assign **I** type when there exists a neighboring cell with $\Phi > 0$ or **L** otherwise.

Here we assume that Φ is the level set implicit function [12].

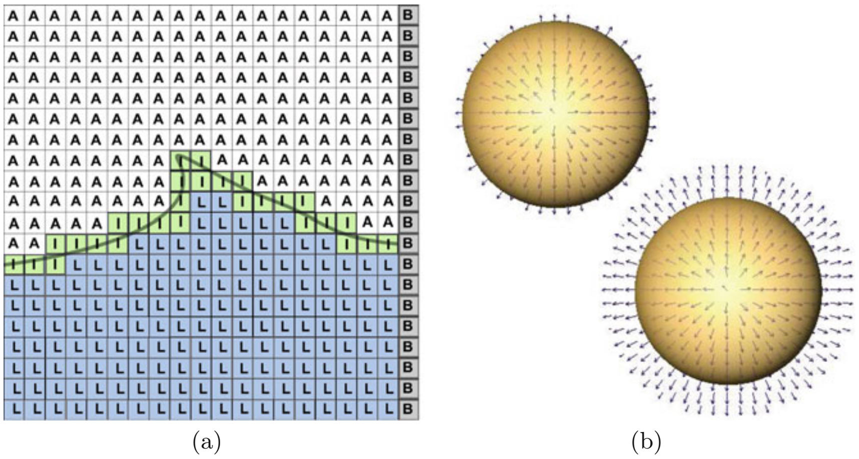


Fig. 2. (a) Cell types: **A** - air phase, **L** - liquid phase, **I** - interface cells, **B** - boundary; (b) A cut-plane of a spherical velocity field initialized inside a sphere (left) and extended in a narrow band outside of the interface (right).

The dependence of cell type on the value of Φ assures that the assigned cell type will not rely upon the order in which cells are processed. This assumption is very important especially for GPU algorithms as race conditions that could appear in the simulation can be very hard to detect.

Defining *interface cells* at the beginning of simulation provides an additional benefit. There is no need to check if neighbors of cells inside the volume exist during Φ evaluation. Therefore, the number of *if* branches is minimal implying high speed of the kernel execution.

Velocity Extension. The velocity field generated by the LBG stream-collide process is valid only inside the bulk of liquid. For a proper numerical advection of the level set and marker particles, velocity field needs to be defined on both sides of the interface. That is why, an iterative velocity extrapolation method is employed in the simulation. Calculation of the velocity field values takes place only in cells that are located in the region external to the liquid. The reason is that the velocity field is defined inside the liquid by the stream-collide step. The length of the iteration process is fixed in the simulation because numerically correct values are needed only inside a narrow band of the interface. An example of velocity field extension inside a narrow band outside the interface is presented in Fig. 2(b).

Level Set Advection. Advection of the level set function Φ is implemented using the first-order forward Euler method in time and an upwind differencing of Φ values. Numerical errors induced by the first-order accuracy are reduced by use of the marker particles that correct the level set function values at each iteration.

Particle Advection. Particle Advection equation is solved using the second order Runge-Kutta method - the midpoint method. The velocity field has to be interpolated in particle positions because marker particles are not bound to the discrete lattice nodes and can move freely inside the lattice boundaries. We use linear interpolation in each dimension. After the first step of the midpoint method a check is performed if the particle left the lattice boundaries. If so, the particle is marked as to be excluded from all computations for the rest of simulation.

Level Set Correction. After the level set advection, calculated by means of the low-order numerical scheme, the zero level set isosurface will be distorted. Therefore, information about the position of marker particles is used to correct the values close to the interface. The correction is performed with the Gaussian kernel base function [11]. The use of the weight function with the Gaussian kernel leads to a simple correction equation. Choosing the kernel radius ensuring that the weight function does not vanish only in a close proximity of $\mathbf{x}$, allows for bounding summations only to particles located in the close vicinity of $\mathbf{x}$. For each active particle, neighboring cells are selected and the particle impact on every cell is stored.

In the Marker Level Set implementation on a GPU proposed in [10], the correction of the level set values by marker particles was achieved by means of a shader and volume rendering. In this paper a different approach is proposed. The level set values correction is performed in two successive steps:

1. Every particle's impact on the closest 27 cells ($3 \times 3 \times 3$ cube) is computed and stored in a temporary array. A problem can occur in a situation when two or more particles affect one cell. If these particles are processed in parallel on the GPU, a race condition may occur that would make computations invalid. Therefore, an atomic addition CUDA function *atomicAdd* is used for storing computed weight impact values.
2. For each cell of the level set grid, total weight is taken from the temporary array and a correction value is subtracted from the original level set value. Once again *atomicAdd* function is used to prevent possible race conditions.

Level Set Reinitialization. After the correction phase, level set values located on the zero isosurface have proper values. The level set values that are far from the interface are still distorted by low-order advection scheme. For that reason, the level set reinitialization step is performed. The Fast Iterative Method [19] is used. The number of iterations is fixed because for computations only level set values from the narrow band of the zero level set are required.

Particle Adjustment. It must be guaranteed that before the next iteration all active particles will be located on the zero level set. As this constraint may have been violated during the level set reinitialization an adjustment process is executed. The particles that are outlying are moved in the direction normal to the interface by an arbitrary fraction of the distance to the interface. A natural choice would be to set this fraction to 0, so that the particle advection always defines the interface. Unfortunately, this leads to numerical artifacts caused by large distances between the interface and particles for turbulent modes. Therefore, the parameter defining this fraction has to be determined experimentally for a specific simulation.

Cells Refilling. Every change of a type of cell may lead to a situation where an air cell becomes partially filled with fluid or a fluid cell becomes empty. The level set function automatically handles the case of an empty cell occurrence. On the other hand, in case of a partially filled cell we need special handling. Change from an empty cell to an interface cell leads to a problem where the stream-collision process would operate on a cell for which the distribution function has no values. For that reason, after such event, cell's distribution function is reinitialized. The new value is equal to the equilibrium distribution function with arguments averaged on all non-empty neighbors. Density ρ and velocity $\mathbf{v}$ are averaged and a distribution function is computed for each cell that changed types from **A** to **I** (Fig. 2(a)). It is assumed that there is no possibility of a change from **A** to **L**.

Streaming. The copy operation of distribution function values between cells is the basic procedure of the streaming step. The distribution function is double buffered and there is only one write operation for each of the distribution function values. Therefore, there is no possibility of a race condition. The 18 assignments (we use the D3Q19 velocity set [13]) are executed in a straightforward way. However, a different approach needs to be taken for liquid streamed from interface cells. Interface cell by definition has at least one empty neighboring cell. This means that the distribution function value for a direction opposite to the direction of the neighboring empty cell will not be assigned during the streaming process. To handle this case, the distribution function reconstruction procedure [7] is employed. Streaming from fluid and interface cells is separated from streaming from solid boundary cells to keep branching cost at minimum. Streaming from solid cells implements the *no-slip* boundary conditions and handles lattice boundary cases by means of *if* conditionals.

Collision. The collision step is an inherently local operation, i.e., it does not use information from other neighboring cells. As a result, very efficient implementation on a GPU is possible. The collision is performed only in cells that have non empty distribution functions. That is, only liquid and interface cells are processed. To increase the stability of simulation, the Smagorinsky sub-grid model [14] is implemented. As was said before, collision process uses only information from the current cell.

3 Results

We performed two test simulations for the common problems of a liquid drop falling to a liquid surface and a breaking dam. The snapshots from the simulation of a liquid droplet are shown in Fig. 3. One can clearly see the moment of the topological change during merge of the drop with fluid surface. The initial splash and the secondary droplet are not resolved in the simulation due to lack of the surface tension in the model. This could however, be easily incorporated by extending the model by introducing the force dependent on the curvature. The comparison of performance for a GPU and simple CPU implementations is presented in Table 1. All of the tests were performed on NVIDIA Fermi based GeForce GTX 460 board and standard AMD Athlon II X4 635 processor. The GPU-based engine outperforms the single core CPU-based implementation an order of magnitude. The snapshots from simulation of the breaking dam problem can be seen in Fig. 4. As shown in the figure, the container is partially filled with liquid. The liquid section is separated from the rest of container volume by an obstacle. Initially the liquid is stationary, then, at a given moment, the obstacle is removed and the liquid is collapsing freely by means of the gravitational force. The impact of the *no-slip* boundary conditions is apparent in the first few snapshots, where the fluid is slowed down by the walls. Despite a large number of

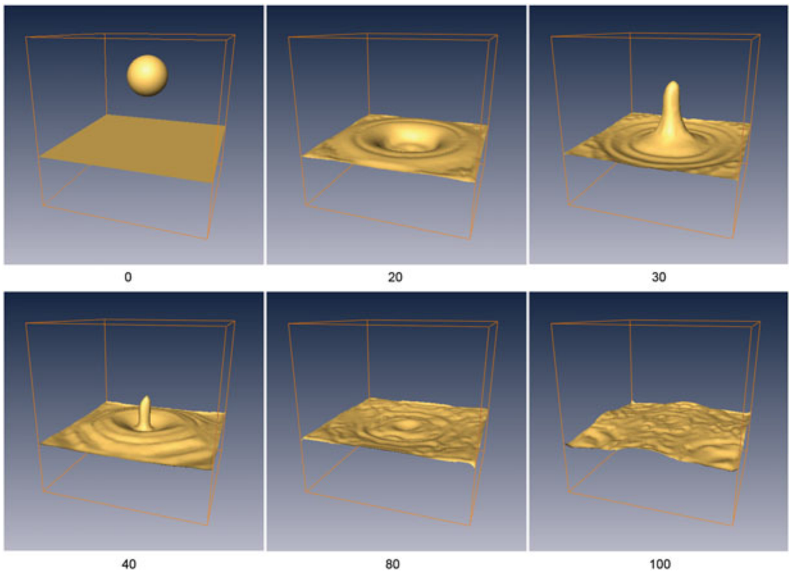


Fig. 3. The snapshots from simulation of a falling drop. Numbers represent the iteration of the simulation.

Table 1. Performance comparison of CPU and GPU based implementations. Values in the table present execution times of one iteration in milliseconds.

	CPU		GPU	
	Average	Std. dev.	Average	Std. dev.
Falling drop	4322.14	1445.86	437.86	39.33
Breaking dam	3765.66	1387.11	651.01	101.78

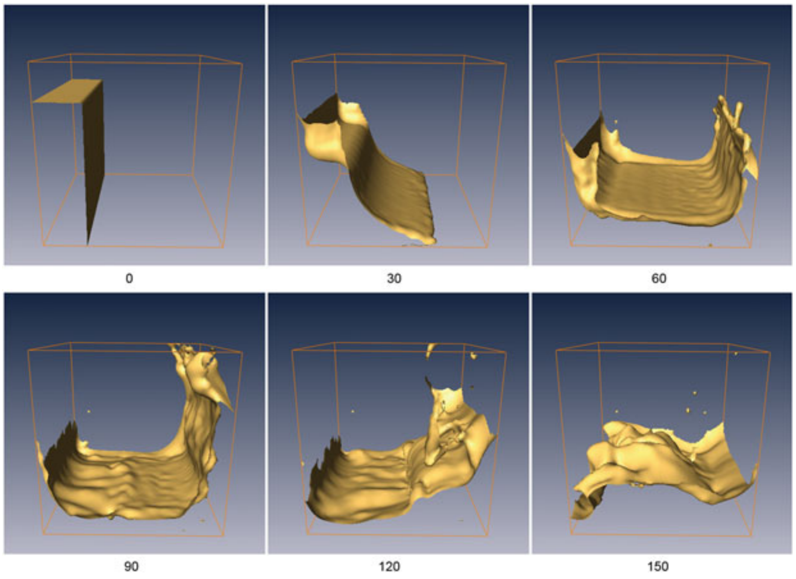


Fig. 4. The snapshots from breaking dam simulation. Numbers represent the iteration of the simulation.

iterations a loss of mass is negligible. On the last snapshots small liquid droplets can be seen falling. This effect is resolved thanks to the marker particles that can preserve fine fluid volume details.

4 Related Work

The application of the LBG approach for simulating free surface flow is dictated by the straightforward mapping of the lattice streaming and collisions onto the GPU computation model. Other discrete CFD methods, including molecular dynamics (MD) [1,3], dissipative particle dynamics (DPD) [4,5] and smoothed particle hydrodynamics (SPH) can also be used for modeling free surface flow. However, all of these methods are rather computationally demanding and require additional mechanisms to control the interface dynamics [1,3,5].

The approach of coupling LBG with level sets for free surface dynamics simulations similar to that presented above has been introduced in [16]. However, unlike out method, it suffers from nonphysical loss of mass. No implementation details about the hardware architecture are given in [16] but it seems that the method targets serial CPU architecture without any parallelism. The LBG/LS simulation engine presented in this paper conserves mass due to the utilization of marker particles. Additionally, the GPU version of the engine performs far better in terms of performance than its single-core CPU counterpart.

Another similar method coupling LBG with the Particle Level Set method (PLSM), called Hybrid lattice Boltzmann/level sets method (HLBM) is also described in [8]. It reports an improved performance compared to the original PLSM [6]. However, it is still behind the performance of our method. The Marker Level Set method used in our approach involves less particles to capture fine interface details and, therefore, performs better in comparison with HLBM. The hybrid of the level set interface tracking and the Particle Level Set method with the SPH is another successful approach used for simulating free surface flow. In [9] visually appealing results are presented demonstrating free-surface simulations capturing fine details of the flow such as sprays. However, the integrated models of SPH and LS are very computationally demanding due to particle motion changing constantly the nearest neighbors of SPH particles. Their efficient implementation on GPU is very difficult and give unconvincing benefits.

5 Conclusions

In this paper we propose a new concept, which can be applied for efficient simulation of a free surface evolution. It integrates the lattice Boltzmann gas and the level set simulation methodologies. The inherent parallelism of LBG allowing for optimal use of GPU architecture together with the possibility of control of the liquid/gas interface by the level sets make the method competitive to other known CFD approaches. Promising performance of its GPU/CUDA implementation demonstrates that the method could be successfully adopted in different areas of physical simulations such as multiple-phase flows, flames spreading, shock and detonation waves tracking and others. Due to its high efficiency, our approach could be potentially used by game and simulator designers. However, interactive visualization of free surface flows still needs faster GPU processors, more work on LBG parallelization and more efficient coupling schemes with level set methods. To be more physically correct, the model should be extended e.g., by including surface tension. Summarizing, the concept presented is very promising and deserves more attention in the future.

Acknowledgements. This research is supported by the AGH Faculty of Computer Science, Electronics and Telecommunications, Dean's Student Grant. We thank Mr. Maciej Kluczny for his essential contribution to this paper.

References

1. Alda, W., Dzwinel, W., Kitowski, J., Moscinski, J., Pogoda, M., Yuen, D.A.: Complex fluid-dynamical phenomena modeled by large-scale molecular-dynamics simulations. Comput. Phys. **12**(6), 595–600 (1998)
2. Anderson, J.D.: Computational Fluid Dynamics: The Basics with Applications. McGraw-Hill, Inc., New York (1995)
3. Dzwinel, W., Alda, W., Pogoda, M., Yuen, D.A.: Turbulent mixing in the microscale. Physica D **137**, 157–171 (2000)
4. Dzwinel, W., Yuen, D.A.: Dissipative particle dynamics of the thin-film evolution in mesoscale. Mol. Simul. **22**(6), 369–395 (1999)
5. Dzwinel, W., Yuen, D.A.: Rayleigh-Taylor instability in the mesoscale modeled by dissipative particle dynamics. Int. J. Mod. Phys. C **12**(1), 91–118 (2001)
6. Enright, D., Losasso, F., Fedkiw, R.: A fast and accurate semi-Lagrangian particle level set method. Comput. Struct. **83**, 479–490 (2005)
7. Korner, C., Thies, M., Hofmann, T., Thurey, N., Rude, U.: Lattice Boltzmann model for free surface flow for modeling foaming. J. Stat. Phys. **121**, 179–196 (2005)
8. Kwak, Y., Nakano, A.: Hybrid Lattice-Boltzmann/level-set method for liquid simulation and visualization. Int. J. Comput. Sci. **3**(579), 1–14 (2009)
9. Losasso, F., Talton, J., Kwatra, N., Fedkiw, R.: Two-way coupled SPH and particle level set fluid simulation. IEEE Trans. Visual Comput. Graph. **14**(4), 797–804 (2008)
10. Mei, X., Decaudin, P., Hu, B.G., Zhang, X.: Real-time marker level set on GPU. In: International Conference on Cyberworlds, CW '08, September 2008, pp. 209–216. IEEE, Hangzhou (2008)
11. Mihalef, V., Sussman, M., Metaxas, D.: The marker level set method: a new approach to computing accurate interfacial dynamics. J. Comput. Phys. (2007)
12. Osher, S., Fedkiw, R.: Level Set Methods and Dynamic Implicit Surfaces. Applied Mathematical Sciences. Springer, New York (2003)
13. Rubinstein, R., Luo, L.S.: Theory of the lattice Boltzmann equation: symmetry properties of discrete velocity sets. Phys. Rev. E **77**(3), 036709 (2008)
14. Smagorinsky, J.: General circulation experiments with the primitive equations. Mon. Weather Rev. **91**(3), 594–595 (1963)
15. Succi, S.: The Lattice Boltzmann Equation for Fluid Dynamics and Beyond. Clarendon Press, Oxford (2001)
16. Thuerey, N., Ruede, U.: Free surface lattice-Boltzmann fluid simulations with and without level sets. In: Proceedings of the Vision, Modelling, and Visualization, VMV, pp. 199–207 (2004)
17. Tolke, J., Krafczyk, M.: TeraFLOP computing on a desktop PC with GPUs for 3D CFD. Int. J. Comput. Fluid Dyn. **22**(7), 443–456 (2008)
18. Wesseling, P.: Principles of Computational Fluid Dynamics. Springer Series in Computational Mathematics. Springer, Berlin (2009)
19. Jeong, W.-K., Whitaker, R.T.: A fast iterative method for a class of Hamilton-Jacobi equations on parallel systems. University of Utah Technical report UUCS07010, pp. 1–25 (2007)

Creation of Agent's Vision of Social Network Through Episodic Memory

Michał Wrzeszcz[1(✉)] and Jacek Kitowski[1,2]

[1] Department of Computer Science, Faculty of Computer Science, Electronics and Telecommunications, AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Krakow, Poland

[2] ACC Cyfronet AGH, AGH University of Science and Technology, ul. Nawojki 11, 30-950 Krakow, Poland

{wrzeszcz,kito}@agh.edu.pl

Abstract. Human societies appear in many types of simulations. One of the most important and the most difficult society elements to be modelled is the social context. In this paper we show how social context can be provided using agents that are equipped with internal visions of social relations between others. Internal vision is a representation of social relations from the agent's point of view so, being subjective, it may be inconsistent with the reality. We introduce the agent model and the mechanism of rebuilding the agent's internal vision that is similar to that used by humans. An experimental proof of concepts is also presented.

Keywords: Social networks · Behaviour modelling · Simulation of human societies · Multi-agent systems · Social context

1 Introduction

Simulations of human societies may be used to create virtual worlds that allow to predict collective behaviour of a crowd. However, these simulations of human societies are useful only if they are highly realistic - the behaviour of simulated individuals must be as similar as possible to human behaviour. This article does not focus at any particular behaviour so we define realism as a replication or imitation of human decision-making process where each human is independent but may be influenced by external factors including other people.

Classically, the simulation of individuals requires definition of 4 main ingredients [5,18,20]: high-level behaviour, perception, animation and graphics. Not all simulations require advanced visualization so the third and fourth aspects will not be considered in this article. To model perception of individuals we can adopt software agents interacting with the environment [15] thus, in this article, we treat each individual as a software agent. A great challenge in all simulations is high-level behaviour modelling. Research proved that the agents based on PECS model (Physical conditions, Emotional state, Cognitive capabilities and Social status [17]) and ontology can be configured to simulate specific scenarios [10].

User reputation in social network is also a field of study (e.g. [6]). Furthermore, Nazir, Prendinger and Seneviratne showed [14] that their pattern based mobility model reproduces well day-to-day human activities of people. However, the difficulty of modelling of high-level behaviour increases significantly when behaviour of a simulated individual depends on social context. For this reason, we focus our research on social context provision. Nevertheless it should be noticed that issue of context in simulations is very wide and this article considers context understood as influence of one individual on another.

In this article we present the model that provides social context by equipping each agent with its internal, conceivably subjective, vision of social relations between others. The internal vision is a representation of social relations from the agent's point of view so it may be inconsistent with the reality. We also show that defining, in an appropriate way, functions that update agent's internal vision of social relations on the basis of observations of interactions of other agents allows the agent to build its internal vision in similar to human way (the social network built by the introduced algorithms is similar to the social network built by human manually on the basis of observed people questioning).

The rest of the paper is organized as follows. In Sect. 2, we show various methods of social context provision, indicating their shortcomings. An extension of the classic agent model is proposed in Sect. 3. Next, in Sect. 4 we verify our model experimentally. Afterwards, we conclude this paper in Sect. 5.

2 Provision of Social Context

Several social context models have been introduced. In [13] a model that describes how agents influence each other within one organization is shown. The basis of the model are beliefs and organizational code. The beliefs are elements of the agent's internal representation of the world while organizational code is a string of values that represents the organization's approximation of beliefs about that reality. In each period, every agent alters any given belief to conform to that of the organizational code with some probability, reflecting the rate of socialization of individuals in the organization. The organizational code also alters any given belief based on the dominant belief of the set of agents - the superior group, defined as those agents whose individual beliefs correspond better with reality than does the code's. The model was later extended by Kane and Prietula [7]. They let individuals learn from (be influenced by) one another. The probability that a given individual would learn from the other individuals rather than the code was represented by an additional parameter. However, it should be noticed that the agents easily develop erroneous beliefs [3].

Another interesting approach is the tag-based computational model [2], which uses tags to describe agents features. In this model, only similar agents can influence each other. Similarity of agents is calculated by comparison of their tags.

One of the latest approaches that use virtual world to show how people behave in particular situations was developed by the EUSAS project (European

Urban Simulation for Asymmetric Scenarios) [9,11]. In this project, the social influence of agents, that were divided into groups, was modelled according to the Latane formula of strength, immediacy and number of other agents [12]. The agent's internal state was changed on the basis of observation of actions of other agents that belong to the same group, e.g., aggression of the agent was growing when it observed aggressive actions of other agents that belonged to the same group. The strength of this effect depended on social position of the observed agent.

The presented models are not able to reproduce some phenomena from the real word where a group of people often contains pairs of close friends as well as pairs of people who do not know each other personally. The influence of a good friend is often higher than influence of other persons even with higher social position, hence the social context model should contain a social network, which describes relations between each pair of agents. Nevertheless, a single social network may be not enough. In the real world, attitude to people depends on past interactions and information received from others. When somebody is not known personally, our attitude to them depends on their relations with people we know and his behaviour that we can observe. Unfortunately, we cannot always evaluate relations between other people correctly. If we observe argue of two friends, we can assume subjectively that they do not like each other. Each human being usually has his own interval vision of social relations that link people around so each agent should have its own internal vision of the social network.

After equipping each agent with its internal vision of the social network we can easily improve the social context models. For instance, similar model like in the EUSAS project may be used, however, the strength of the influence could depend on strength of the social relation (in internal social network) between observer and observed agent, not obligatorily on the social position of observed one. Therefore, the key issue is provision of mechanism that updates agent's internal vision of the social network.

3 Agent Model

The proposed agent model uses the classical approach. The classical approaches introduce a deliberative agent [5,20] that contains a symbolic model of the world and in which decisions are made via logical reasoning, based on pattern matching and symbolic manipulation. The agent also gathers information about past events. Storing this type of information is called the episodic memory. It was introduced by Vere and Bickmore [19] in the agent called HOMER.

The agent model is shown in Fig. 1. The agent observes the environment using sensors and stores information about important events in the memory. In our case, the agent stores information about possible interactions of other agents. On the basis of this information it builds its internal representation of the world, which describes social relations of known individuals. The agent uses this internal vision of environment, together with information about itself, to plan its actions (it is done by logic engine) that are executed by effectors.

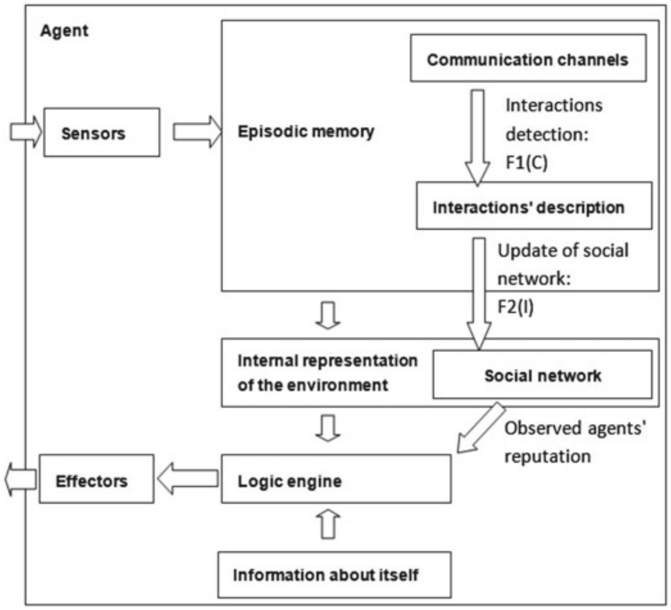

Agent
Sensors
Episodic memory
Communication channels
Interactions detection: F1(C)
Interactions' description
Update of social network: F2(I)
Internal representation of the environment
Social network
Observed agents' reputation
Effectors
Logic engine
Information about itself

Fig. 1. Agent model

The internal social network may be multi-layered [8] to describe various types of relations, e.g., professional relations, friendship etc. The Multi-layered Social Network is defined as a tuple $< V; E; L >$ where: V is a non-empty set of nodes, E is a set of edges and L is a set of layers [1]. The edge is a tuple $< x, y, l >$, where x, y are different nodes and l is a layer $(x, y \in V, l \in L, x \neq y)$. The layer is a set of relations of the same type (e.g. trust layer, family ties layer). Maximum two relations between particular nodes (x to y and y to x) belong to each layer: $< x, y, l >\in E\wedge < x', y', l' >\in E \wedge x = x' \wedge y = y' \Rightarrow l \neq l'$.

The described agent uses many well-known techniques. The use of sensors, effectors, episodic memory and internal representation of the world is not new. Translation of the social network to the form useful for the logic engine has also been described, e.g., in [16] a mechanism of calculation of social reputation on the basis of social relations was described. Reputation of the agent may be used by the logic engine to verify reliability of messages sent by it. Thus, we propose a novel method of building and updating of agent's internal vision of social network what is, in our opinion, a key to increasing realism of the simulations.

The update of the social network is done through the episodic memory. The agent observes the environment and represents observed social events as a set of communication channels. Representation of the environment as the set of communication channels was described in [21]. The agent creates the communication channel in its memory when it observes the possibility of communication between pair of other agents and destroys the channel when it observes that this pair is not able to communicate any longer. The communication channel contains information about the period of time in which described agents were able to communicate. It may also contain some additional information if available, e.g., attitude of one observed agent to another evaluated on the basis of its gestures. The communication channels are analysed by function $F1$ to provide interactions' description. The $F1$ function decides if the channel describes interactions or not. If the channel describes interactions, the function assigns a type to them. Afterwards, on the basis of the interactions' types, the relations in the social network are updated using function $F2$. The interaction's type is used to choose which layer of network should be updated and decides how value of the relation in chosen layer should be changed, e.g., observation of kind gestures should increase the value of friendship relation while observation of argue should decrease this value. More formal definitions of $F1$ and $F2$ are presented below:

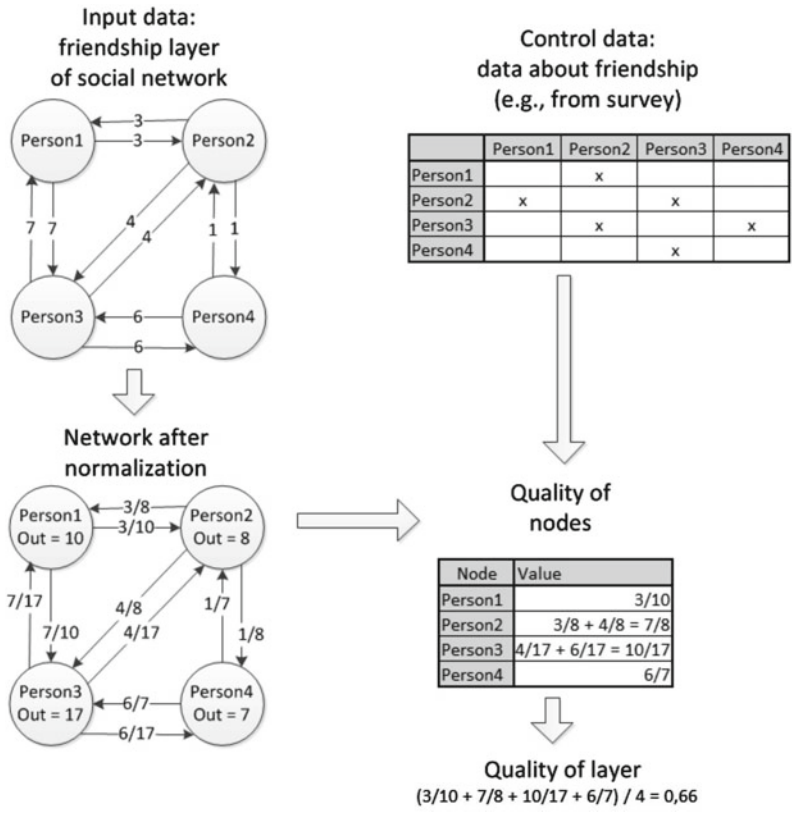
- $F1(C) \rightarrow \{I1, I2\} \vee NULL$ where:
  - $C = \{A1, A2, D, O\}$ - communication channel between pair of agents where:
    $A1$, $A2$ - agents connected by channel,
    D - duration of the channel,
    O - other information about observed event if available (e.g., attitude of agents $A1$ and $A2$),
  - $\{I1, I2\}$ is a tuple that describes interactions - $I1$ represents actions of $A1$ while $I2$ actions of $A2$; function may return $NULL$ when event described by the communication channel is not an interaction (e.g., when one agent walked by another and they did not see each other); the $I1$ and $I2$ interactions may be different when additional information is included in the communication channel (e.g., if $A1$ attacks $A2$ and $A2$ only defends itself, $I1$ will represent aggressive interaction initialized by $A1$ while $I2$ not aggressive response of $A2$); interaction is defined as a tuple $I = \{A1, A2, T\}$ where:
    $A1$, $A2$ - agents involved in interaction,
    T - type of interaction,
  - $NULL$ - ignore this communication channel this time, no interactions are produced (function F2 will not be used),
- $F2(I) \rightarrow [U]$ where:
  - I - interaction,
  - $[U]$ is a list that describes updates of the social network that should be done on the basis of the interaction; function returns a list because updates of more than one layer may be performed on the basis of one interaction. Update is a tuple $U = \{A1, A2, L, V\}$ where:
    $A1$, $A2$ - agents between which relation should be updated,
    L - layer of social network that should be updated,
    V - the amount of change of relation (numerical value).

Sample functions $F1$ and $F2$ are shown in the next section.

The quality of each layer of the social network may be evaluated using the following formulas:

– calculate output value of each node:
  - $node\_out\_value = \sum_{outgoing\_relations} relation\_strength$,
– normalize the relations' strengths:
  - $normalized\_strength = \frac{relation\_strength}{start\_node\_out\_value}$,
  - $start\_node\_out\_value$ is the $node\_out\_value$ of node from which the relation starts,
– calculate quality of node:
  - $node\_quality = \sum_{outgoing\_relations\_included\_in\_control\_data} normalized\_strength$,
  - control data contains relations that indeed exist in the real world (the social network created by the agent is subjective while the control data is objective),
– calculate quality of layer:
  - $layer\_quality = \frac{\sum_{all\_nodes} node\_quality}{number\_of\_nodes}$.

The quality of layer equal to 1.0 means that this layer describes only relations included in the control data so the obtained results fit perfectly to the reality



	Person1	Person2	Person3	Person4
Person1		x		
Person2	x		x	
Person3		x		x
Person4			x	

Node	Value
Person1	3/10
Person2	3/8 + 4/8 = 7/8
Person3	4/17 + 6/17 = 10/17
Person4	6/7

Fig. 2. Evaluation of friendship layer in the exemplary social network

as evaluated independently by the individuals. Evaluation of friendship layer in the exemplary social network is shown in Fig. 2. The input data and the control data were artificially created to illustrate the algorithm.

4 Tests

The aim of creation of the introduced agent model was the increase of realism of individuals' behaviour in simulations with independent agents being influenced by external factors including other agents. To verify it we compare the social network built using the proposed approach with that constructed manually by human on the basis of the data collected in the real world. Results conformity would verify ability of the approach to mimic well some human processes during decision making. Hence, the verification of functions ($F1$ and $F2$) usage for creation of two separated layers of the social network - one showing friendship and another professional relations of a group of observed people - were performed.

During experiments we used the Reality Mining Dataset [4] that included data collected in the real world. The Reality Mining Dataset incorporates the 94 subjects that had completed the survey conducted in January 2005 . Of these 94 subjects, 68 were colleagues working in the same building on campus (90 % graduate students, 10 % staff) while the remaining 26 subjects were incoming students at the university's business school.

For our experiments, we have used three types of data included in the dataset. The first type was Bluetooth data from subjects' telephones. Using MAC addresses of the Bluetooth devices discovered on each Bluetooth scan and times of Bluetooth scans we could reproduce possible interactions of subjects. The second type was a survey where each subject indicated his friends. This data was used to verify if the constructed friendship layer was correct. The third type was survey data that described which subjects see each other every day in office. This data was used to identify coworkers of each subject.

4.1 Experiment 1

In our experiment, the social network was evolving over the time, from the first time of Bluetooth scan to the last time of Bluetooth scan. Communication channels were updated at each time of scan. Additionally, at each time of scan, the interactions were identified (use of $F1$ functions) and the social network was updated (use of $F2$ function). Simple $F1$ and $F2$ functions were used. Function $F1$ was creating a pair of interactions when a communication channel between agents existed at the time of scan. The type of each interaction was "general". For each interaction, function $F2$ was incrementing the value of relation from $A1$ to $A2$ (see definition of $F2$ in previous section) in both layers if relation existed. If not, the relation was created with value one. After last Bluetooth scan, qualities of layers have been calculated using instruction described in Sect. 3. The control data for friendship layer contained relations between people that marked each other in survey as a friend while the control data for coworkers layer contained relations between people identified as coworkers (on the basis of survey). The quality of friendship layer equal to 1.0 means that this layer describes only relations between friends, the quality of coworkers layer equal to 1.0 means that this layer describes only relations between coworkers.

The quality of friendship layer was 0.18 while the quality of coworkers layer was 0.5 (see Table 1 in Sect. 4.2). This is consistent with expectations because people spend more time at work than with friends, and this method treated all meetings equally.

4.2 Experiment 2

Experiment 2 was similar to experiment 1. We have only redefined $F1$ and $F2$ functions on the basis of our everyday experience. People meet friends mainly in the evenings during weekends while coworkers are met during office hours from Monday to Friday. Additionally, meeting duration should be taken into account. Meeting with close friends is usually longer than meeting with people we know but they are not our friends. Moreover, during office hours, time that we spend near coworkers is usually very long - much longer than the time of visit of a person that has only some business to us. Therefore we have defined $F1$ and $F2$ functions as follows:

- $F1$ - create a pair of interactions between subjects connected by communication channel when following requirements are fulfilled:
  - If communication channel has existed last 10 scans, and the day of week is Saturday or Sunday, and the time is between 9 and 12 pm., create friends' interactions.
  - If communication channel has existed last 60 scans, and the day of week is not Saturday or Sunday, and the time is between 8 am. and 4 pm., create coworkers' interactions.
- $F2$ - update value of relation between pair of subjects participating in interaction when following requirements are fulfilled:
  - If interaction type is friends' interaction, increment value of relation from $A1$ to $A2$ (see definition of $F2$ in previous section) in the friendship layer if the relation exists. If not, create it with value one.
  - If interaction type is coworkers' interaction, increment value of relation from $A1$ to $A2$ (see definition of $F2$ in previous section) in the coworkers layer if the relation exists. If not, create it with value one.

Results of the experiment are shown in Table 1. The quality of friendship layer was 0.80 while the quality of coworkers layer was 0.95. It means that the quality of friendship layer is more than four times better than in the experiment 1 and the quality of coworkers layer is almost two times better.

However, it should be noticed that the functions $F1$ and $F2$ proposed by us may not be optimal. The aim of the experiment was not to find the best $F1$ and $F2$ functions but to show that the proposed solution works. Probably more complicated functions would give better results but even such simple functions showed that introduced technique works well and can be used to simulate societies.

Table 1. Results of the experiment

Layer	Layers' qualities	
	Experiment 1	Experiment 2
Friendship	0.18	0.80
Coworkers	0.5	0.95

5 Conclusion

We have shown that we are able to a build reliable social network only on the basis of observation of interactions of others. The experiment proved that defining, in an appropriate way, functions $F1$ and $F2$, results in successful identification of various types of social relations. The proposed agent model increases the realism of the social simulations because simulated individuals have their internal vision of social relations exactly like humans (agents can have different opinion about relations that link other individuals in the system because different people may have different opinions about social relations that link people around). Moreover, introduced mechanism of the update of this internal vision has been successfully verified using test data from the real world.

Presented model may be used in simulations of human societies where modelling of influence of one individual on others is important issue e.g. it can be used to create simulations for police that allow to check if a demonstration can turn into a riot. Computer games, especially role-playing games (RPGs), are other important application of the model. In RPGs artificial intelligence controls non-player characters (NPCs). The presented algorithms can be used to model NPCs and allow them to identify their friends and opponents. In the near future we want to test behaviour of the introduced agents in environments similar to role-playing game.

Acknowledgments. This research is partially supported by AGH-UST grants no. 11.11.230.015 and 15.11.230.097.

References

1. Bródka, P., Filipowski, T., Kazienko, P.: An introduction to community detection in multi-layered social network. In: Lytras, M.D., Ruan, D., Tennyson, R.D., Ordonez De Pablos, P., García Peñalvo, F.J., Rusu, L. (eds.) WSKS 2011. CCIS, vol. 278, pp. 185–190. Springer, Heidelberg (2013)
2. Chen, Y., Prietula, M.: To deceive or not to deceive? Mimicry, deception and regimes in tag-based models. In: Intra-Organizational Networks (ION) Conference (2005)
3. Doran, J.: Social simulation, agents and artificial societies. In: Third International Conference on Multi-Agent Systems, pp. 4–5 (1998)
4. Eagle, N., Pentland, A., Lazer, D.: From the cover: inferring friendship network structure by using mobile phone data. Proc. Natl. Acad. Sci. U S A **106**, 15274–15278 (2009)

5. Genesereth, M., Nilsson, N.: Logical Foundations of Artificial Intelligence. Morgan Kaufmann, San Mateo (1987)
6. Han, Y.S., Kim, L., Cha, J.W.: Computing user reputation in a social network of web 2.0. Comput. Inf. **31**, 447–462 (2012)
7. Kane, G., Prietula, M.: Influence and structure: extending a model of organizational learning. In: Twelfth Annual Organizational Winter Science Conference (2006)
8. Kazienko, P., Brodka, P., Musial, K., Gaworecki, J.: Multi-layered social network creation based on bibliographic data. In: SocialCom/PASSAT, pp. 407–412. IEEE Computer Society (2010)
9. Kryza, B., Krol, D., Wrzeszcz, M., Dutka, L., Kitowski, J.: Interactive cloud data farming environment for military mission planning support. Comput. Sci. **23**(3), 89–100 (2012)
10. Kvassay, M., Hluchy, L., Kryza, B., Kitowski, J., Seleng, M., Dlugolinsky, S., Laclavk, M.: Combining object-oriented and ontology-based approaches in human behaviour modelling. In: 2011 IEEE 9th International Symposium on Applied Machine Intelligence and Informatics (SAMI), pp. 177–182 (2011)
11. Laclavík, M., Dlugolinský, Š., Šeleng, M., Kvassay, M., Schneider, B., Bracker, H., Wrzeszcz, M., Kitowski, J., et al.: Agent-based simulation platform evaluation in the context of human behavior modeling. In: Dechesne, F., Hattori, H., ter Mors, A., Such, J.M., Weyns, D., Dignum, F. (eds.) AAMAS 2011 Workshops. LNCS, vol. 7068, pp. 396–410. Springer, Heidelberg (2012)
12. Latane, B.: Dynamic social impact. In: Hegselmann, R., Mueller, U., Troitzsch, K.G. (eds.) Modelling and Simulation in the Social Sciences from the Philosophy of Science Point of View, vol. 23, pp. 287–310. Springer, Berlin (1996)
13. March, J.G.: Exploration and exploitation in organizational learning. Organ. Sci. **2**(1), 71–87 (1991)
14. Nazir, F., Prendinger, H., Seneviratne, A.: Participatory mobile social network simulation environment. In: ICC, pp. 1–6. IEEE (2010)
15. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, Englewood Cliffs (1995)
16. Sabater, J., Sierra, C.: Reputation and social network analysis in multi-agent systems. In: AAMAS '02: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 475–482. ACM (2002)
17. Schmidt, B.: Modelling of human behaviour: the PECS reference model. In: 14th European Simulation Symposium (2002)
18. Thalmann, D.: Simulating a human society: the challenges. In: Computer Graphics International, CGI02, pp. 25–38 (2002)
19. Vere, S.A., Bickmore, T.W.: A basic agent. Comput. Intell. **6**, 41–60 (1990)
20. Wooldridge, M., Jennings, N.R.: Intelligent agents: theory and practice. Knowl. Eng. Rev. **10**(2), 115–152 (1995)
21. Wrzeszcz, M., Kitowski, J.: Mobile social networks for live meetings. Comput. Sci. **13**(4), 87–100 (2012)

The Influence of Multi-agent Cooperation on the Efficiency of Taxi Dispatching

Michał Maciejewski[1,2](✉) and Kai Nagel[2]

[1] Institute of Machines and Motor Vehicles, Faculty of Machines and Transportation, Poznan University of Technology, Ul. Piotrowo 3, 60-965 Poznan, Poland
michal.maciejewski@put.poznan.p

[2] Transport Systems Planning (VSP), Institute for Land and Sea Transport Systems, TU Berlin, Salzufer 17-19 Sekr. SG12, 10587 Berlin, Germany
{maciejewski,nagel}@vsp.tu-berlin.de

Abstract. The paper deals with the problem of the optimal collaboration scheme in taxi dispatching between customers, taxi drivers and the dispatcher. The authors propose three strategies that differ by the amount of information exchanged between agents and the intensity of cooperation between taxi drivers and the dispatcher. The strategies are evaluated by means of a microscopic multi-agent transport simulator (MATSim) coupled with a dynamic vehicle routing optimizer (DVRP Optimizer), which allows to realistically simulate dynamic taxi services as one of several different transport means, all embedded into a realistic environment. The evaluation is carried out on a scenario of the Polish city of Mielec. The results obtained prove that the cooperation between the dispatcher and taxi drivers is of the utmost importance, while the customer–dispatcher communication may be reduced to minimum and compensated by the use of more sophisticated dispatching strategies, thereby not affecting the quality of service.

Keywords: Dynamic taxi dispatching · Dynamic vehicle routing · On-line optimization · Multi-agent simulation · MATSim · Traffic flow simulation · Simulation-based optimization

1 Introduction

Taxi dispatching is one of the most crucial components of operational taxi fleet management, however the research in this area is limited. There are several other problems in the realm of operational research that are similar to taxi dispatching, such as the *Dynamic Single Load Pickup and Delivery Problem* [5], also known as the *Real-time Multivehicle Truckload Pickup and Delivery Problem* [13], the *On-line Dial-a-Ride Problem* [2], or the *Demand-Responsive Transport*-oriented problems [6]. However, the (partial) independence of drivers, restricts the adaptation of the existing optimization methods mainly to those cases when a taxi fleet is managed centrally (e.g. [7,12]), which is not common.

Sophisticated dynamic routing approaches are hard to analyse theoretically (e.g. competitive analysis), and therefore simulation tools have to be used. In transport-related problems, simulation has to incorporate realistically modelled dynamism of customer demand, traffic flow phenomena and fleet management operations. These aspects are even more crucial when considering urban areas due to high dynamics of traffic flow resulting in continuously changing travel times and often, depending on the type of services, in high volatility of demand (e.g. taxi).

Particularly in the case of taxi dispatching, the use of microscopic traffic simulators allows to evaluate the performance of the service under different, often extreme, scenarios, such as sport/cultural events, bad weather conditions or public transport strikes [7,11]. Furthermore, the (partial) independence of taxi drivers can be addressed by multi-agent simulation [1,4,11]. These issues, however, remain almost unexplored. To the best knowledge of the authors, the only application of both microscopic traffic simulation and multi-agent simulation to a real-life scenario has been carried out by Seow et al. in Singapore [11].

To sum up, we think that the best approach is to use multi-agent traffic simulation that allows for running large-scale scenarios at the microscopic level of detail. Out of various simulation platforms considered, MATSim [3] is arguably the one that is closest to meet all the requirements stated; see [8] and references therein for additional justification. The base dynamic taxi dispatching problem and its implementation has been described in [9]. This paper aims at providing insight into possible ways of modelling and simulating multi-agent cooperation in taxi dispatching in MATSim and assessing the efficiency of different cooperation schemes that have been modelled.

2 Cooperation in Taxi Dispatching

Providing efficient taxi services requires proper collaboration between the main actors, that is customers, taxi drivers and the dispatcher. There are various models of providing the services, each assuming a different level of collaboration between these actors. In the simplest one, customers order a taxi without informing about their destination, which is given only to taxi drivers after entering the taxi. Moreover, taxi drivers do not notify the dispatcher about the status of the current request until it is completed. On the opposite pole, one may consider a taxi service where customers provide the dispatcher with their destinations, taxi drivers are monitored on-line, and orders may be dynamically reassigned between taxi drivers in reaction to the current situation. One may expect that the more collaboration between the actors occurs, the more efficient the service is. However, to investigate this issue thoroughly, one may use multi-agent simulation that would mimic the interaction between actors, and between actors and the environment, such as urban traffic.

Within the current research, the following options have been considered:

- *destination knowledge* – the destination is known a priori if a customer informs the dispatcher about his/her destination.

– *request execution monitoring* – the dispatcher may monitor taxis and constantly update the timing of their schedules, which is necessary if a longer scheduling horizon is considered. Otherwise, taxi drivers notify the dispatcher only about switching between the *busy* and *idle* states.
– *requests reassignment* – already assigned requests can be dynamically reassigned between drivers. Request swapping is expected to be beneficial for both customers and drivers, and is usually coordinated by the dispatcher.

Enabling any of these properties implies some extra collaboration between interested parties. The first one involves additional customer-to-dispatcher communication, the second one imposes extra driver-to-dispatcher communication, while the last one requires real-time collaboration between drivers and the dispatcher.

3 Platform for Simulation of Taxi Services

In order to carry out the research, an integrated simulation platform has been developed [9]. The system consists of two fundamental components, namely MATSim [3] and the DVRP Optimizer [8]. The former is used for modelling transport supply (including a taxicab fleet) and demand (including taxi demand) and providing queue-based traffic flow simulation. The latter is responsible for managing a fleet of taxis (or in general, any vehicle fleet) within the simulation.

Since both components are integrated tightly, one can simulate taxi services dynamically (MATSim), where events, such as request submissions, vehicle departures or arrivals, trigger optimization or update procedures (the DVRP Optimizer). Taxi drivers are modelled as dynamic (reactive) agents that move in a city network, together with other drivers, and communicate with the dispatcher and customers. Each taxi driver has a schedule made up of the following task objects:

– `DriveTask` – driving along a given route (usually the shortest path between two points in the network).
– `ServeTask` – picking-up a passenger at a given location (includes waiting for the passenger).
– `WaitTask` – waiting at a given location for a new customer.

When a taxi customer wants to take a taxi, he/she calls the taxi service (time T_i^0; the order is registered as request i) and waits until the taxi arrives. In response, the taxi dispatcher assigns the new request to one of taxis, according to a given algorithm. At time T_i^1, the selected taxi sets off for the customer. The taxi arrives at the pick-up location at time T_i^2, and after the customer is picked up, it departs (time T_i^3). Finally, the taxi drops off the passenger at the destination location at time T_i^4. At this moment, the taxi driver may start out for the next request or wait. Because of the stochasticity of taxi demand and traffic flow, times T_i^1, T_i^2, T_i^3 and T_i^4 are subject to change during simulation. The correspondence between the taxi schedule and the respective customer's plan is presented in Fig. 1.

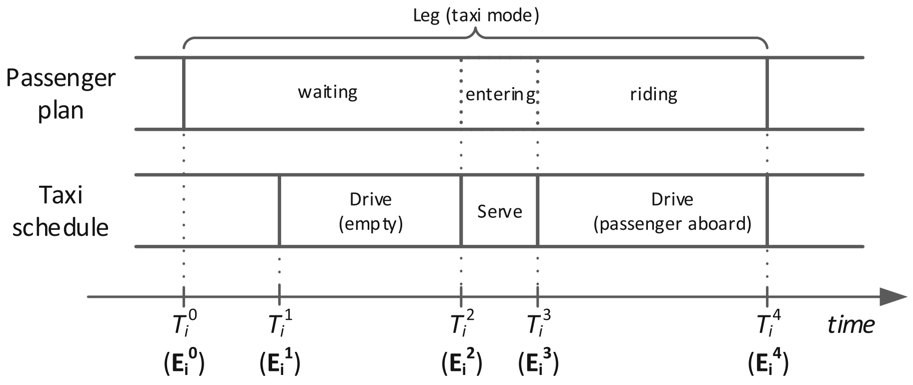


Fig. 1. A planned taxi leg and the corresponding sequence of taxi tasks

4 On-line Taxi Dispatching Algorithms

Taxi dispatching deals with dynamic and stochastic demand and supply. Typically, a dynamic optimization algorithm reacts to various changes, represented as events, that occur over time. The simplest approach, commonly used by taxi companies, consists in responding to submissions and completions of requests (events $\mathbf{E_i^0}$ and $\mathbf{E_i^4}$, respectively). To increase efficiency, the algorithm can be triggered also when taxis set off for, arrive at and depart from pickup locations (events $\mathbf{E_i^1}$, $\mathbf{E_i^2}$ and $\mathbf{E_i^3}$, respectively). On the opposite pole, all taxicabs can be monitored on-line so that the algorithm can be executed in case any taxi is even slightly ahead of/behind schedule. In this research, it has been assumed that vehicles are not monitored and the algorithm responds only to events $\mathbf{E_i^0}$–$\mathbf{E_i^4}$.

Since on-line dispatching procedures must be time efficient, fast local optimum search methods, or even just local update methods, have to be used instead of global optimization. Provided that these procedures are fast enough (respond almost instantly to events), we can assume that they operate on static data, which are considered a kind of snapshot of the current system state.

Since customers perform immediate taxi calls and then wait for a taxi to come, minimization of the total waiting time is the optimization objective. To assure fairness of the dispatching process, all taxi requests are prioritized according to their submission time and scheduled based on the *first-come, first-served* policy.

Three different strategies, namely no-scheduling, one-time scheduling and re-scheduling, have been implemented. The first one assumes minimal communication between drivers and the dispatcher. The next one monitors request execution, while the last one extends the second one by adding the functionality of reassigning requests. All of them can be applied both with or without the a priori destination knowledge.

No-scheduling strategy (NOS). This strategy reacts to the following events:

- $\mathbf{E_i^0}$ – the *nearest*[1] vehicle among the idle ones is dispatched to this request; if no vehicle is available at that time, the request is queued in a FIFO queue
- $\mathbf{E_i^4}$ – the vehicle that has just completed request i is dispatched to the first request in the FIFO queue; otherwise, the vehicle becomes idle

This strategy imitates the way orders are assigned to taxis in a typical taxi company. The main advantage is low demand for computational power. Moreover, this strategy does not require travel times to be known since it does not build schedules; one can even use straight-line distance to find the nearest idle taxi. The drawback is that its performance deteriorates as the number of idle taxis decreases — if all taxis are busy, the first idle one may appear on the opposite side of a city. The a priori destination knowledge is not taken into account.

One-time-scheduling strategy (OTS). This strategy updates the existing taxi schedules by appending a new request to the schedule of the nearest vehicle among all vehicles (both idle or busy), where *nearest* again typically means "nearest in time". Therefore, the knowledge of travel times is mandatory. This strategy monitors execution of requests and constantly updates the timelines of schedules, but without request reassignment. The strategy acts in the following way:

- $\mathbf{E_i^0}$ – request i is appended to the schedule of the *nearest* taxi
- $\mathbf{E_i^1}$ – $\mathbf{E_i^4}$ – if the vehicle serving request i is ahead of/behind time, the timing of its schedule is updated, while the assignments remain unchanged

This strategy considers all the available vehicles, not only the idle ones, which broadens the choice of taxis and thus increases the chances of finding a better assignment. However, when destinations are unknown, the planning horizon is limited up to one pick-up ahead (event $\mathbf{E_i^3}$), and therefore, vehicles with already one planned pick-up cannot be considered when scheduling a new request. The weakest point of OST is the permanence of assignments, even if a vehicle is seriously delayed.

Re-scheduling strategy (RES). This strategy is an enhanced version of the previous one. The difference is that requests may be re-assigned between taxis if the other one appears to be *nearer*. The strategy is defined as follows:

- $\mathbf{E_i^0}$ – request i is appended to the schedule of the *nearest* taxi
- $\mathbf{E_i^1}$ – $\mathbf{E_i^4}$ – if the vehicle serving request i is ahead of/behind time, full rescheduling is carried out; the assignments are subject to change

In consequence, the scheduling algorithm runs in two modes: on the arrival of a new request (exactly as in case the OTS strategy), and when one of taxicabs is reported to be ahead of/behind schedule (all planned request are re-scheduled). RES overcomes the weaknesses of the previous ones, but at the cost of extra computational time.

[1] Depends on the distance measure chosen; usually means 'nearest in time'.

5 Test Scenario

The computational analysis was carried out for Mielec, a city in south-eastern Poland, with a population of over 60’000 inhabitants. The model was derived from a macroscopic model of private transport in Mielec, used as a small-size test instance in several studies, e.g. [10]. The network model consists of over 200 nodes and 600 links. The whole study area is divided into 13 zones; nine of them represent city districts, while the rest – external areas. Based on the original model that described afternoon 1-hour peak, the demand data have been artificially generated to cover the period between 6:00 am and 8:00 pm, resulting in over 42’000 private transport trips that represent a hypothetical case of day traffic, including both morning and afternoon rush-hour traffic.

The performance of the proposed optimization strategies was tested against different variants of the scenario (each repeated 20 times). The taxi demand was modelled as 3, 5 and 7 % of the intra-urban private transport demand (i.e. the number of orders n equalled 917, 1528 or 2175, respectively), while the taxi fleet size m was 50 and 100. Since the fleet size was constant over the entire simulation and the taxi demand peak coincided with the private transport demand peak, the rush hours were the most challenging for taxi dispatching.

6 Simulation Results

Different performance measures were used during the simulation studies, all described in [9]. They represented either the customers’ or taxi company’s point of view, often mutually conflicting. The selected ones are:

- average passenger waiting time, $T_{\mathrm{W}} = \sum_{i \in N}(T_i^2 - T_i^0)/n$
- maximum passenger waiting time, $T_{\mathrm{W}}^{\max} = \max_{i \in N}(T_i^2 - T_i^0)$
- average pickup trip time, $T_{\mathrm{P}} = \sum_{i \in N}(T_i^2 - T_i^1)/n$

Figures 2, 3, 4 show the results obtained for different m and n. Separate curves were plotted for the taxi fleet size of 50 (n/m between 18.34 and 43.5) and of 100 (n/m between 9.17 and 21.75). The following notation is used:

- *NOS* – NOS (destination knowledge n/a)
- *OTS w/o D* – OTS *without* the destination knowledge
- *OTS w/D* – OTS *with* the destination knowledge
- *RES* – RES both *with* and *without* the destination knowledge (aggregated due to negligible differences)

Figure 2 shows, in terms of T_{W}, that RES and OTS give comparable results for low and medium load. However, as the system gets overloaded, OTS *without* the destination knowledge behaves similarly to NOS, producing random assignments as the search space gets reduced to only 1 taxi (explained below). These relations are clearly visible in the experiments with 50 taxis and demand equal to 5–7 %, where periods of high load alternate with ones of lower load. In these

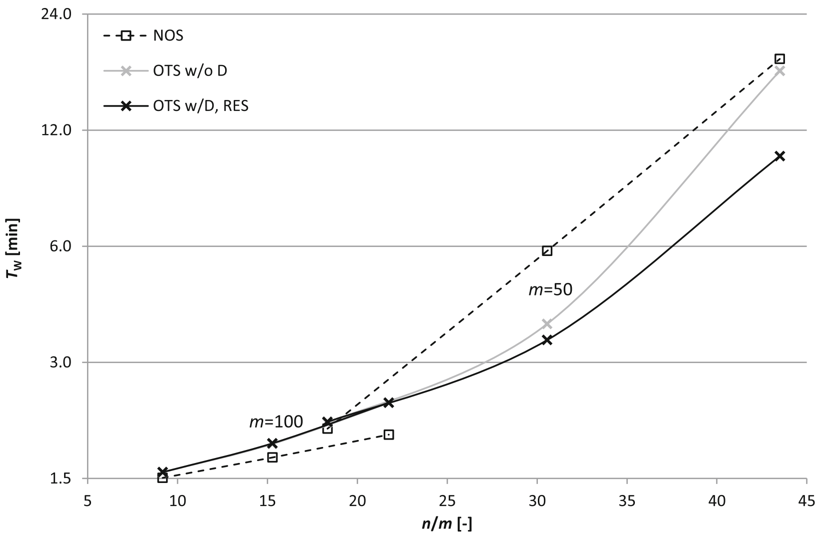


Fig. 2. Average passenger waiting time T_{W} at different demand-supply ratios

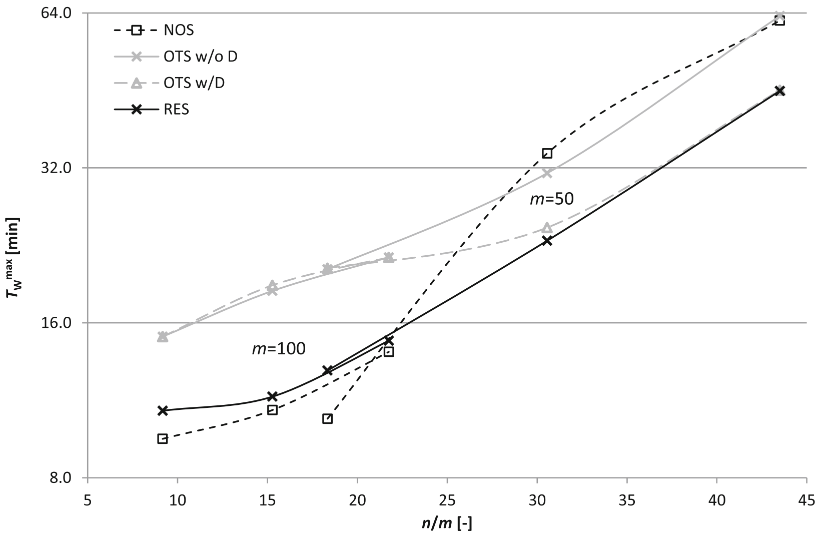


Fig. 3. Maximum passenger waiting time $T_{\mathrm{W}}^{\max}$ at different demand-supply ratios

cases, OTS *without* the destination knowledge performs similarly either to NOS or to RES, thereby giving in-between outcomes.

The behaviour of OTS *without* the destination knowledge at high loads is caused by the fact that when a taxi with customer i aboard departs (event $\mathbf{E_i^3}$) it notifies the dispatcher about the current destination, and therefore, the

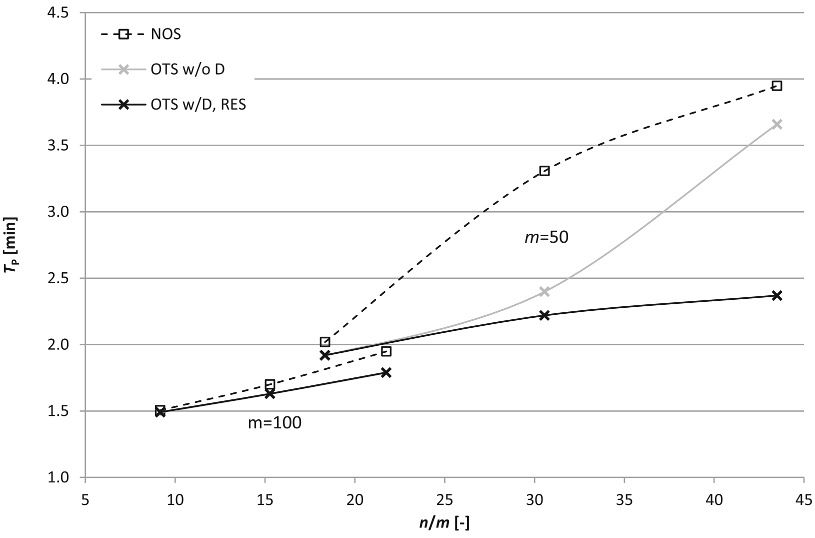


Fig. 4. Average pickup trip time T_{P} at different demand-supply ratios

dispatcher can schedule the next event, $\mathbf{E_i^4}$, and then append customer j (the first from the queue) to this taxi's schedule. This resembles the behaviour of NOS, with the exception that in the latter, the assignment is done after $\mathbf{E_i^4}$, not $\mathbf{E_i^3}$. Actually, making a random assignment following $\mathbf{E_i^3}$ is even worse since it adds more variability in terms of the waiting time ($T_j^2 - T_j^0$ depends on $T_i^4 - T_i^3$ now). As a result, the behaviour gets less fair (some customers must wait considerably longer than others), which is proved by larger $T_{\mathrm{W}}^{\max}$ (see Fig. 3).

Another interesting result is that the NOS series in Fig. 2 are not adjacent. This is due to the fact that the average distance to the closest idle taxi is proportional to $1/\sqrt{m_{\mathrm{idle}}}$, where m_{idle} is the number of idle taxis. Therefore, T_{W} in experiment *3 %:50* (i.e. 3 % demand and 50 cabs) is about $\sqrt{2}$ times higher than in *3 %:100*, and slightly higher compared to *7 %:100*.

A quite surprising outcome, in terms of T_{W}, is the poorer performance of RES and OST in experiments with 100 taxis, compared to NOS. This is caused by a biased arrival time estimator for on-going trips that is used for scheduling. This estimator ignores the fact that the longer the trip lasts the more probable that it will end later than previously expected, and consequently, gives too optimistic estimates (discussed in [9]). As a result, the OTS and RES strategies give preference to non-idle vehicles, which results in shorter pickup trips (see Fig. 4), but at the cost of longer taxi awaiting.

Figure 3 shows the maximum waiting times, $T_{\mathrm{W}}^{\max}$, that may be interpreted as the upper bound (or a first-guess estimate) of T_{W} during morning and afternoon peaks. In-depth analysis of $T_{\mathrm{W}}^{\max}$ reveals that in the low load regime, NOS is the most fair strategy, while OTS and RES suffer from the biased estimation. However, wrong decisions made by RES can be reversed to some extend by later reassignments of requests, thereby reducing the performance gap. On the other hand, at high n/m ratios, the randomness of NOS and OTS *without* the destination knowledge results in higher $T_\mathrm{W}^\mathrm{max}$.

Figure 4 represents the taxi company's perspective. Although the main motivation of using OTS and RES is the reduction of passenger waiting times, they have a beneficial side effect of minimizing the operating costs. They both yield lower T_P since by considering also non-idle vehicles, they not only minimize T_i^2 (and thus T_W) but also allow for $T_i^1 > \tau$, which reduces T_P even further. The discontinuity of the 50-taxi and 100-taxi series is caused by the fact that $T_\mathrm{P} \sim 1/\sqrt{m_\mathrm{idle}}$, as explained earlier.

Similarly to the case of T_W, RES and OTS give comparable T_P for low and medium load. However, in an overloaded system, OTS *without* the destination knowledge operates similarly to NOS, while OTS *with* the destination knowledge performs similarly to RES. Lower T_P implies more idle taxis at the dispatcher's disposal, hence the dearth of taxis during the high peak is shorter and less acute.

7 Conclusions

The main goal of the paper was to propose and evaluate various schemes of collaboration between agents. The platform integrating microscopic, behaviour-based traffic simulation (MATSim) and dynamic vehicle routing optimization (DVRP Optimizer) allowed to realistically model, simulate and evaluate dynamic taxi services for the scenario of the Polish city of Mielec.

Out of three different optimization strategies, each assuming a different collaboration scheme between agents, that is customers, taxi drivers and the dispatcher, the best results have been obtained for the RES strategy. RES imposes the highest level of cooperation between taxi drivers and the dispatcher. In particular, at high demand-to-supply ratios, it turns out to be very effective, both for customers (lowest T_W and $T_\mathrm{W}^\mathrm{max}$) and the company (lowest T_P). On the other hand, in the low load regime, all three strategies perform similarly well. Interestingly, NOS shows more preference towards customers, while OTS and RES offer more benefit to the company. This issue is related to the biased travel time estimation, and will be addressed in future by adding on-line vehicle monitoring. With this enhancement, the RES strategy is expected to outperform the other ones regardless of the demand-supply relation.

Another interesting outcome is the limited use of the a priori destination knowledge, which implies that the dispatcher may respect customers' privacy and not ask them about their destinations. This information is not used by NOS, and has a negligible impact on RES. Only in the case of OTS, the not-knowing the customer's destination deteriorates taxi dispatching since contrary to RES, wrong decisions cannot be reversed. However, the lack of this knowledge may be compensated by the use of RES, which is superior, or at least equal, to OTS in every respect.

To conclude, the cooperation between the dispatcher and taxi drivers is of the utmost importance, while the customer–dispatcher communication, if necessary, may be reduced to minimum. The lack of the knowledge of destination can be balanced out by means of more sophisticated dispatching strategies so that the quality of service remains unaffected. Among plans for the future is the extension of the platform functionality with an on-line vehicle tracking module in order to simulate the real-time driver–dispatcher cooperation. Another aim is to extend taxi drivers' independence by letting them to follow their own *cruise-or-wait* strategies when idle.

References

1. Alshamsi, A., Abdallah, S., Rahwan, I.: Multiagent self-organization for a taxi dispatch system. In: 8th International Conference on Autonomous Agents and Multiagent Systems. pp. 21–28 (2009)
2. Ascheuer, N., Krumke, S.O., Jörg, R.: Online dial-a-ride problems: minimizing the completion time. In: Horst, R., Sophie, T. (eds.) STACS 2000. LNCS, vol. 1770, pp. 639–650. Springer, Heidelberg (2000)
3. Balmer, M., Meister, K., Rieser, M., Nagel, K., Axhausen, K.: Agent-based simulation of travel demand: structure and computational performance of MATSim-T. In: Innovations in Travel Modeling (ITM) '08, Portland, Oregon, June 2008, also VSP WP 08-07. www.vsp.tu-berlin.de/publications (2008)
4. Cheng, S., Nguyen, T.: Taxisim: a multiagent simulation platform for evaluating taxi fleet operations. In: Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 02, pp. 14–21. IEEE Computer Society (2011)
5. Fleischmann, B., Gnutzmann, S., Sandvoß, E.: Dynamic vehicle routing based on online traffic information. Transp. Sci. **38**(4), 420–433 (2004)
6. Horn, M.: Fleet scheduling and dispatching for demand-responsive passenger services. Trans. Res. Part C Emerg. Technol. **10**(1), 35–63 (2002)
7. Lee, D., Wang, H., Cheu, R., Teo, S.: Taxi dispatch system based on current demands and real-time traffic conditions. Transp. Res. Rec. J. Transp. Res. Board **1882**(–1), 193–200 (2004)
8. Maciejewski, M., Nagel, K.: Towards multi-agent simulation of the dynamic vehicle routing problem in MATSim. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011, Part II. LNCS, vol. 7204, pp. 551–560. Springer, Heidelberg (2012)
9. Maciejewski, M., Nagel, K.: Simulation and dynamic optimization of taxi services in MATSim. VSP Working Paper 13-05, TU Berlin, Transport Systems Planning and Transport Telematics. www.vsp.tu-berlin.de/publications (2013)
10. Piatkowski, B., Maciejewski, M.: Comparison of traffic assignment in VISUM and transport simulation in MATSim. Transport Problems (2013, in press)
11. Seow, K., Dang, N., Lee, D.: A collaborative multiagent taxi-dispatch system. IEEE Trans. Autom. Sci. Eng. **7**(3), 607–616 (2010)
12. Wang, H., Lee, D., Cheu, R.: PDPTW based taxi dispatch modeling for booking service. In: Fifth International Conference on Natural Computation, 2009, ICNC'09, vol. 1, pp. 242–247. IEEE (2009)
13. Yang, J., Jaillet, P., Mahmassani, H.: Real-time multivehicle truckload pickup and delivery problems. Transp. Sci. **38**(2), 135–148 (2004)

Basic Endogenous-Money Economy: An Agent-Based Approach

Ivan Blecic(✉), Arnaldo Cecchini, and Giuseppe A. Trunfio

Department of Architecture, Planning and Design, University of Sassari, Sassari, Italy
{ivan,cecchini,trunfio}@uniss.it

Abstract. We present an agent-based model of a simple endogenous-money economy. The model simulates agents representing individual persons who can work, consume, invent new products and related production technologies, apply for a loan from the bank and start up a business. Through the interaction of persons with the firms, we simulate the production of goods, consumption and labour market. This setting allows us to explore how an endogenous-money economy may build up from scratch, as an emergent property of actions and interactions among heterogeneous agents, once the money is being injected into a non-monetary self-production (or barter) economy. We provide and discuss the results of several computational experiments under three scenarios: (1) with just one firm, (2) with a limited number of firms and abundant workforce, (3) and with unlimited number of firms.

Keywords: Agent-based computational economics · Endogenous-money economy · Heterogeneous agents

1 Introduction

The endogenous-money approach to modelling an economic system assumes that banks create money by making loans to firms, which simultaneously creates equivalent deposits [1,2]. There may of course be constraints on how much money the banks are allowed to create in this manner, for example by way of some reserve constraints. But this approach, in the spirit of Moore [3] and Holmes [4], allows for banks to look for the reserves later *after* they have extended the credit.

Such endogenously created credit money is the point of ignition of all the economic activity as it allows firms to hire workers, start the production, pay wages, dividends and interests, and subsequently have the workers, capitalists and bankers consume the goods produced.

The idea of such a closed monetary economy has raised two puzzles: how are profits possible if firms need to repay loans with interests? [5]; and more in general, is an ever-increasing supply of money the only way to have profits in a steady-state economy?

Keen [1,2] has proposed a model with a set of differential equations which shows on the aggregate level that indeed no such increasing supply of money is necessary.

Based on this approach, we have developed an agent-based model of endogenous-money economy to study its evolution on disaggregated level of individual firms, banks, workers and consumers and to account for possible heterogeneities of products, production technologies, workers' skills and consumption preferences which are an inevitable feature of any real-world economy.

Agent-based computational economics (ACE) is a growing field of economic modelling [6], for a review see [7], and several large scale models have been developed addressing macroeconomic policy issues (see [8] for a review).

The purpose of our model is not so much to support policy design starting from a initial scenario populated with persons, firms and banks calibrated on a real-world economy. Rather, what we wanted to explore is if and how an endogenous-money economy may build up from scratch, as an emergent property of actions and interactions among heterogeneous agents, once the money is being injected into a non-monetary α self-production (or barter) economy.

This is how the remainder of the paper is organised. In the following section we specify the model. Then, in Sect. 3 we present and discuss the results of several computational experiments under different scenarios. Finally, in the concluding Sect. 4 we examine some advantages of the agent-based modelling approach and present plans for future developments.

2 The Model

There are three types of agents in our model: persons, firms and a bank. At the beginning of a simulation, the world is populated only by a set of persons. A *Person* represents an individual who can consume, work for firms and start a business by creating a new firm. A *Firm* employs workers, runs the production and sells the produced goods. At each time step it then pays the wages to workers, the dividends to the person who owns the firm, and the interests on the load to the bank. The *Bank*'s role in the model is to evaluate the business plans for new firms proposed by the persons, and to provide loans to the most promising ones.

In what follows we describe the behaviour, the decision-making procedures and the interactions among the three types of agents.

Production. For simplicity, we assume no physical capital is used in the production, so the only production factor is labour. The workforce is differentiated since each person is initialised with n skills whose values are randomly generated from a normal distribution. Firms produce differentiated consumption goods using technologies each represented by a Cobb-Douglas production function. In particular, given a set of workers each endowed with n skills $s_{w,i}$ $(i = 1, \ldots, n)$, the quantity produced with a specific technology is given by:

$$Q = k \prod_{i \in \{skills\}} \left(\sum_{w \in \{workers\}} s_{w,i} \right)^{\alpha_i} \tag{1}$$

where k and α's are specific parameters of the production technology.

Consumption. We assume a person's utility from the consumption of x_i quantities of n goods is given by the following constant elasticity of substitution (CES) utility function:

$$U(x_1, \ldots, x_n) = \left(\sum_{j \in \{goods\}} a_j x_j^{\rho} \right)^{\frac{1}{\rho}} \tag{2}$$

where the share parameters a for different products and the ρ are variable among persons.

This specification of the consumption utility function is grounded on the assumption that firms may produce differentiated products, which is an important feature of our model.

We assume the consumers are rational utility-maximising agents. Therefore, given a set of available goods, their prices p_i and the available budget B the person decides to spend on consumption during each time step, the utility-maximising consumption bundle is determined by:

$$x_i = B \frac{\left(\frac{p_i}{a_i} \right)^{\frac{1}{\rho - 1}}}{\sum_{j \in \{goods\}} a_j^{\frac{1}{1-\rho}} p_j^{\frac{\rho}{\rho - 1}}} \tag{3}$$

Invention of New Products and Production Technologies. At each time step, there is a probability each person invents a new product and the related production technology. A newly invented product is defined by a randomly generated value which represents the average consumers' utility parameter for that good (i.e. the average value of the share parameter a for that good in the utility function (2)). In other words, if the product is put into production, this value is used as the mean of a probability density function through which we randomly assign individual a for that product to each person. In a similar manner, the invented technology is defined by the randomly generated parameters k and α's of the production function in expression (2). Once the product and its production technology have been invented, the person applies for a loan from the bank.

Ranking of the Applications for Loan. We assume, unrealistically but for the sake of simplicity, that there is just one bank in the system, creating credit money through loans to persons and firms. The bank evaluates the applications for loan by persons and decides which to finance. This evaluation is based on the start-up business plan which provides the quantity produced by one adequately (technology-wise) skilled worker and the product's average utility parameter. The bank finances the most promising business plan by providing the loan to the person who then creates the firm.

Firm's Production and Pricing Decision-Making. Given the general turbulence in the system (new firms get created or go bankrupt, there is a competition among firms for consumers' money and for skilled labour, wages and consumption rates may change, and so on), a price established in the past does not necessarily clears the market, nor a production plan stays optimal for a long time.

Therefore, firms gradually adjust their production plans and product pricing based on past outcomes. To do so, they use an adaptive heuristics based on the gradient method in which they shift the price and the production plan and thus try to "learn" their demand function based on the observed past market responses. We implemented this adaptive heuristics of firms as a two-phase procedure. First, each firm experiments price adjustments from step to step in order to keep the inventory stock below a predefined level. Then, once the inventory has stabilised, the firm attempts to increase its production if the current flows of profits allows it to hire more workers, and vice versa, it reduces the production by laying off workers if the profits run below a certain reserve threshold. A small remark: it is fairly

Labour Market. Given that each production technology is defined in terms of workers' skills (see the expression (2) above), the firms express different demand for workers with different skills. Each firm pays the same wage to all its workers, so in order to decide which persons to hire and at what wage, the firm uses a heuristics to select the most productive workers, technology-wise, with respect to their reservation wages. For already employed persons, the reservation wage is the wage they are currently working for, while for unemployed we establish a baseline minimum wage for which the person is willing to accept a job instead of staying unemployed.

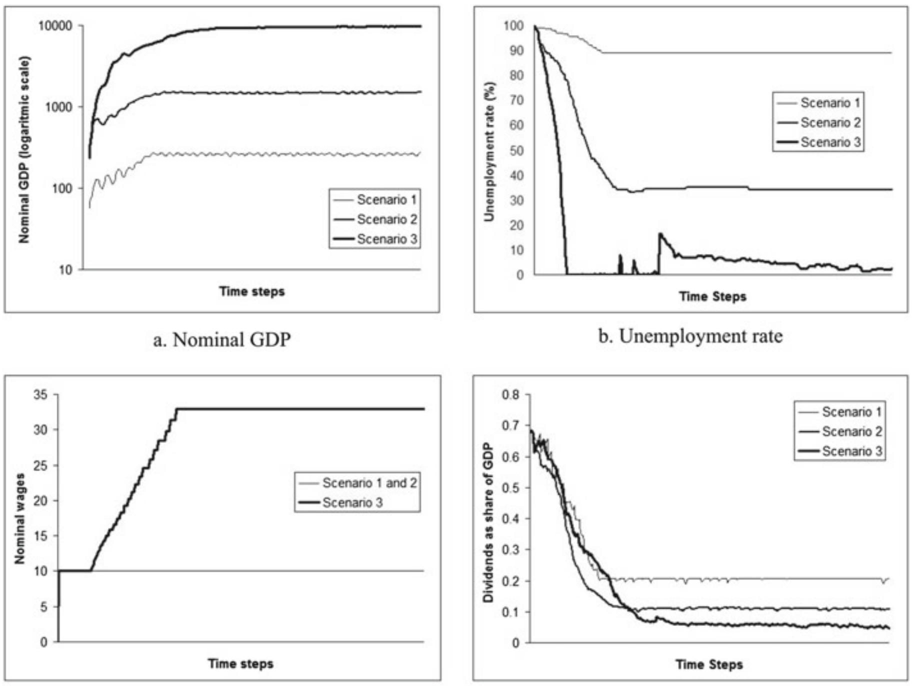
This approach was devised to model the competition for workers among firms. The competition grows as the number of unemployed drops, which is then reflected in the upward wage pressures.

3 Computational Experiments

We present and discuss the results of three computational experiments each ran under a different scenario in relation to the structure of competition among firms and to the relative "abundance" of workforce. All scenarios were initialised with 200 persons and ran for 500 time steps.

The first scenario is a monopolistic setting with a single firm (i.e. we set the limit of only one firm financed by the bank during the simulation).

The second scenario is a more competitive setting with five firms, each with its product and production technology. As we shall see, this setting does not hit the upper bound of absorbing all the available workforce, due to the interplay of the demand and the structure of production costs related to production technologies.



a. Nominal GDP

b. Unemployment rate

c. Nominal wages

d. Dividends as share of GDP

Fig. 1. The evolution of macroeconomic variables of the system under three scenarios: a. Nominal GDP, b. Unemployment rate, c. Nominal wage, d. Dividends as a share of GDP

Finally, the third scenario allows unlimited number of firms. In this case, we'll see that all the available workforce gets absorbed, which in turn toughens the competition among firms for the skilled labour.

Figure 1 shows the evolution of several macroeconomic aggregates. As it was to be expected, the economy in the third scenario, with no limits on the number of firms, produced the greatest overall output (Fig. 1-a). It is important to remember that firms in principle may be created and die at any time step of the simulation. Multiple Monte Carlo runs of the simulation under the third scenario yielded on average a maximum of around 50 firms operating simultaneously at some point in time, but on average only about 30 ended up active at the end of the simulation, while the others were outcompeted and went bankrupt. These numbers are of course sensitive to the model parameters and the decision-making heuristics followed by firms, but they in principle show that given the nature of the production technology and a limited population there is an upper bound of firms that may operate in a steady-state.

The differences among the three scenarios in the overall output are reflected by the unemployment rates (Fig. 1-b). In the case of the monopoly, only a small

Table 1. Firms at the end of the simulation

	Scenario 1	Scenario 2	Scenario 3
Number of firms	1	5	31
Median number of workers	22	28	9
Min. number of workers	22	5	1
Max. number of workers	22	39	19
Median wage	10	10	33

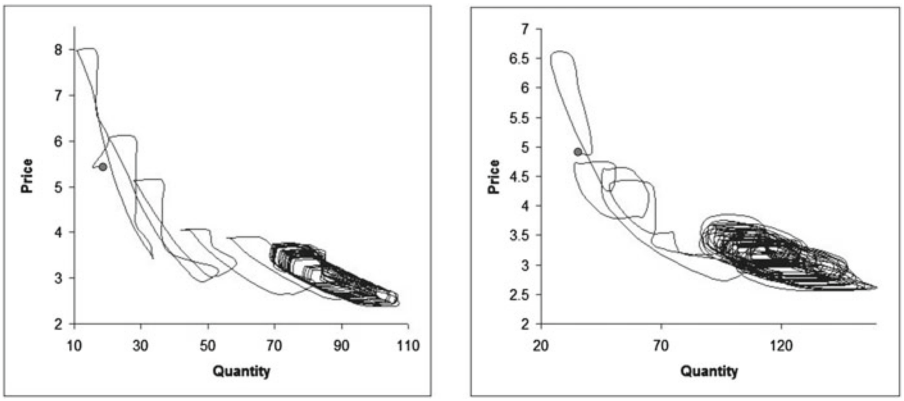


Fig. 2. Firm's behaviour in the price-quantity phase space: prices set by the firm and the quantity of product sold during the simulation (grey dots show the price-quantity at the step of firm's creation). The diagram to the left shows the case of the only firm present in the Scenario 1, to the right the case of one of the five firms in the Scenario 2.

portion – around 10 % – of the available workforce was absorbed by the firm, while in the second scenario 2/3 was employed at the end of the simulation. The economy was able to employ all its production potential only in the third scenario, running the unemployment down close to zero.

What is relevant here to see in combination with the unemployment rates is the wages dynamics (Fig. 1-c). In the first and the second scenario the wages paid to workers remained at the level of the unemployment reservation wage. In the case of the second scenario, this of course is partly due to the fact that, given the internal randomness, in this specific simulation run the five operating firms happened to use sufficiently different production technologies (putting different "weights" on worker skills) in relation to the available pool of skills so as not to bring about competition among firms for "rare" skills. Had it been otherwise, there would have been some upward wage pressure. In any case, nothing of the magnitude observed in the third scenario where the final average wage was more than threefold the unemployment reservation wage.

We don't model rigidities and transaction costs for hiring, laying off and job switching. So, during the simulation under the third scenario we frequently

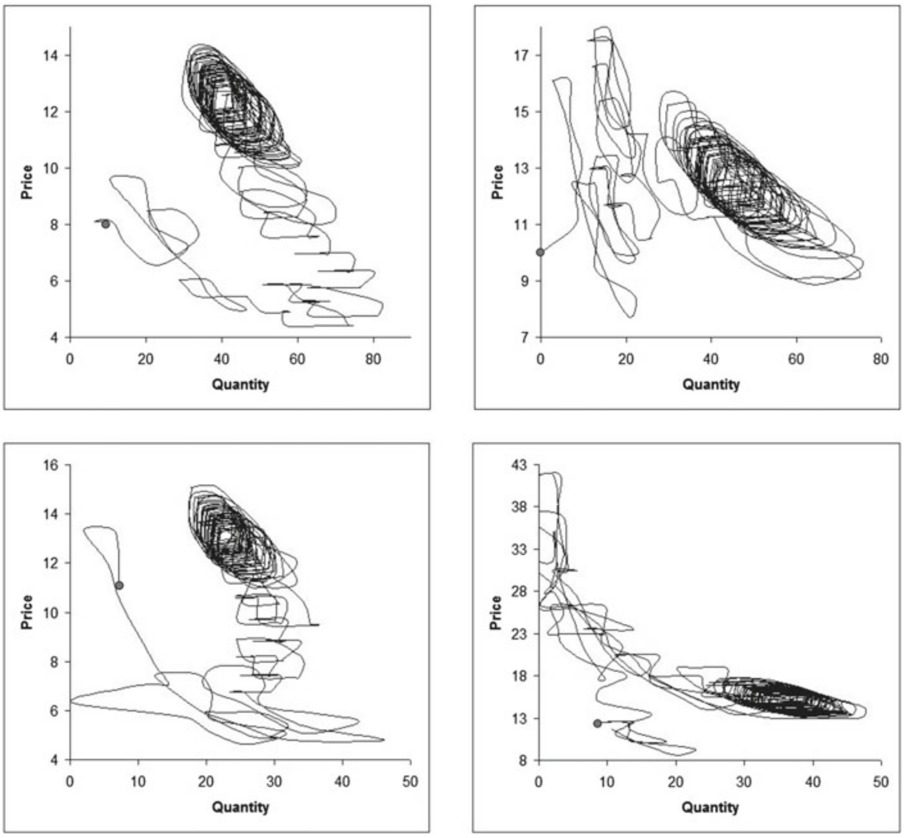


Fig. 3. Prices set and quantity of products sold by four firms in the Scenario 3 during the simulation (grey dots show the price-quantity at the step of firm's creation).

observed highly turbulent intervals of time where workers change jobs from step to step, often back-and-forth among firms trying to outbid the competitors by offering higher wages.

Possibly the most interesting macroeconomic result is told by the Fig. 1-d. It shows how the income is distributed among the firm owners and the workers (and the banker). Here we represent the dividends paid to firm owners as a share of all the incomes of the economy (which is the sum of dividends, wages paid to workers, and interest payments to the banker). We see that in the case of monopoly the firm owner manages to capture the greatest share of the overall income, followed by the second and then the third scenario. Again, the numbers themselves are not as important as the general story they tell.

Let us turn to some micro analysis. In Table 1. we summarise few descriptive statistics of the firm population at the end of the simulation ran under the three scenarios.

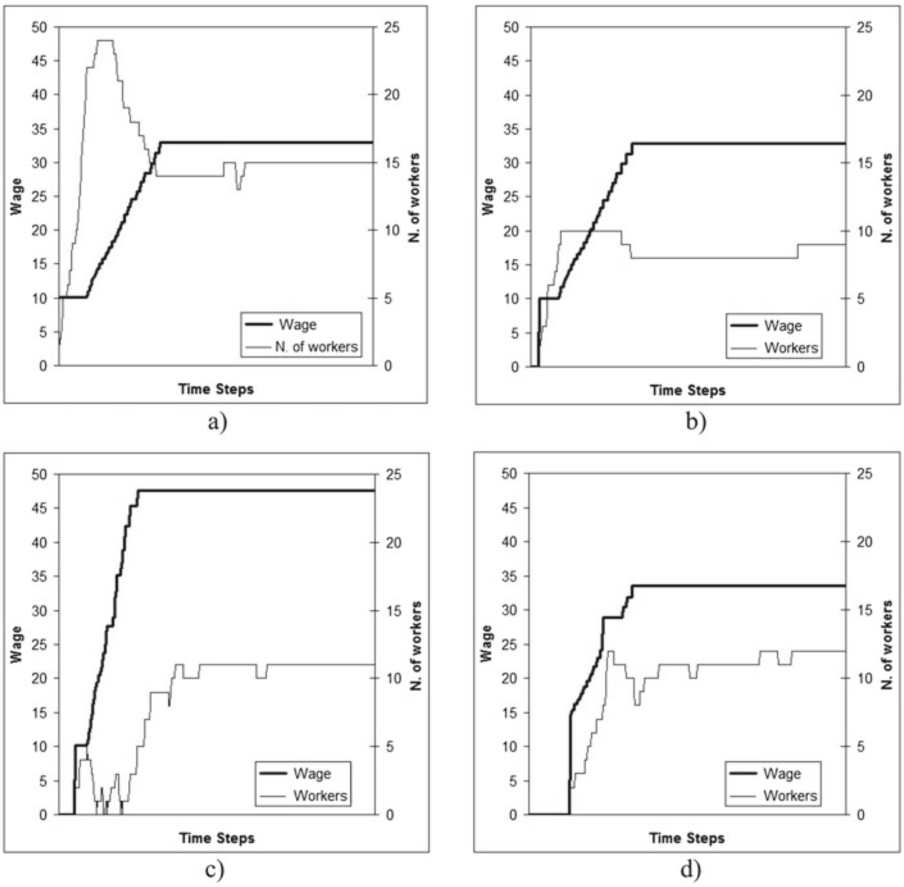


Fig. 4. The number of workers and wages through time steps for four firms in the Scenario 3.

An interesting demonstration of a firm's behaviour is the phase diagram of the prices it sets and the corresponding quantities of the product it manages to sell at those prices. In Fig. 2 we present such phase diagrams for the firm in the Scenario 1 and for one of the five firms in the Scenario 2. They both start from low production levels and settle for high prices. Then, as they expand the production the trajectory gravitates towards lower price levels.

Richer interplays among firms occur when they enter into a stronger mutual competition, as in the Scenario 3. In Fig. 3 we show the phase diagrams for four different firms. Here the trajectories are different, and may depend on the perturbations due to the entrance or bankruptcy of firms and in general their mutual competition.

In Fig. 4 we show the evolution of the wages and the number of workers hired by the same four firms under Scenario 3.

The firm represented in the Fig. 4-a is the first created in the simulation. During an initial phase, it hires the workers at the unemployment reservation wage of 10, and then increases its workforce up to a maximum of 25 workers. Then, once the wages raise beyond the value of 20, it start laying off workers to stabilise at 15 workers. This variations may be due to two distinct effects. One is the rise of wages (due to the competition for workers from other firms) which increases the production costs. The other effect is the product competition from other firms which may negatively influence the demand for the firm's products.

No such magnitude of change in workforce occurred in the firms represented in Fig. 4-b and 4-d. An interesting thing though happened to the firm in Fig. 4-c. As it can be observed in the chart, in two distinct time steps the firm lost all its workforce and was consequently forced to bring its production to a halt. It turns out this was due to the entrance in the market of two other firms with similar production technologies (i.e. highly valuing workers with similar skills) engaging in a fierce wage competition. In those two time steps these two firms actually managed to "capture" all our original firm's workers. Indeed, that there is a differentiated demand for workers skills is proven by the fact that the firm in Fig. 4-c stabilises at a notably higher level of nominal wage. Basically, we are observing a form of labour market segmentation.

4 Conclusions

The results we obtained on the macroeconomic level confirm those of Keen's model [1,2], namely that a constant flow of profits are in principle possible in a steady-state economy without an ever-increasing supply of money. But our agent-based approach to modelling endogenous-money economy have, it seems to us, a few advantages over the aggregate modelling with differential equations and systems dynamics, as it allows several features to arise as emergent properties of the interaction among agents. One notable example is the distribution of the income (and thus of the production surplus) between workers and the firm owners. While this is something that needs to be postulated in the modelling on the aggregate level, it appears instead as an emergent property in our model.

An important distinctive feature of our model is the possibility of products differentiation among firms. This, for instance, is not contemplated in one of the most complete ACE models [9] where no differences in the quality of goods is assumed. Besides the fact this is an notable characteristics of any real economy, some properties exhibited by our model relevant for the economic analysis, emerge precisely because of the assumption of product differentiation among firms.

The model we presented is still quite rudimentary and there are plenty of things we plan to develop in the future. One limiting assumption we make in our model is that the production takes place without physical capital. This greatly simplified our task, for we didn't need to model the production of capital and intermediate goods, nor the procurement of natural resources. This though comes at a cost, because we weren't then able to simulate some features which, we hold, are probably relevant in this context, such as the impact of fixed costs and the related economies of scale.

The financial sector is another area which needs to be wholly developed. Instead of a single bank, we plan to implement multiple banks competing among each other. Finally, in order to make it potentially useful for policy analysis, we'd need to model the government sector and to allow for a more realistic representation of different institutional settings.

References

1. Keen, S.: The dynamics of the monetary circuit. In: Rossi, S., Ponsot, J.F. (eds.) The Political Economy of Monetary Circuits: Tradition and Change, pp. 161–187. Palgrave Macmillian, New York (2009)
2. Keen, S.: Solving the paradox of monetary profits. Economics: The Open-Access, Open Assessment E-Journal, **4**, 2010-2 (2010)
3. Moore, B.J.: The endogenous money supply. J. Post Keynesian Econ. **10**, 372–385 (1988)
4. Holmes, A.R.: Operational contraints on the stabilization of money supply growth. In: Morris, F. (ed.) Controlling Monetary Aggregates, pp. 65–77. The Federal Reserve Bank of Boston, Boston (1969)
5. Rochon, L.P.: The existence of monetary profits within the monetary circuit. In: Fontana, G., Realfonzo, R. (eds.) Monetary Theory of Production: Tradition and Perspectives. Palgrave Macmillian, New York (2005)
6. Farmer, J.D., Foley, D.: The economy needs agent-based modelling. Nature **460**, 685–686 (2009)
7. Tesfatsion, L., Judd, K. (eds.): Agent-Based Computational Economics. Handbook of Computational Economics, vol. 2. North Holland, The Netherlands (2006)
8. Fagiolo, G., Roventini, A.: Macroeconomic policy in DSGE and agent-based models. Working paper series (7), Department of Economics, University of Verona (2012)
9. Cincotti, S., Raberto, M., Teglio, A.: The EURACE macroeconomic model and simulator. In: Aoki, M., Binmore, K., Deakin, S., Gintis, H. (eds.) Complexity and Institutions: Markets, Norms and Corporations. Palgrave Maximillian, London (2012)

Author Index