Parallel Computational Fluid Dynamics

Implementations and Results Using Parallel Computers

A. Ecer
J. Periaux
N. Satofuka
S. Taylor
Editors

North-Holland

Parallel Computational Fluid Dynamics

Implementations and Results Using Parallel Computers

This Page Intentionally Left Blank

PARALLEL COMPUTATIONAL FLUID DYNAMICS

IMPLEMENTATIONS AND RESULTS USING PARALLEL COMPUTERS

Proceedings of the Parallel CFD '95 Conference
Pasadena, CA, U.S.A., 26-29 June, 1995

Edited by

A. ECER
IUPUI
Indianapolis, IN, U.S.A.

J. PERIAUX
Dassault-Aviation
Saint-Cloud, France

N. SATOFUKA
Kyoto Institute of Technology
Kyoto, Japan

S. TAYLOR
California Institute of Technology
Pasadena, CA, U.S.A.

1996
ELSEVIER
Amsterdam – Lausanne – New York – Oxford – Shannon – Tokyo

ELSEVIER SCIENCE B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam, The Netherlands

ISBN: 0 444 82322 0

This book is printed on acid-free paper.

Printed in The Netherlands.

PREFACE

This book contains a collection of papers presented at the Parallel CFD 1995 Conference held on June 26-28, 1995 in Pasadena, California. The Parallel CFD Conference is international in scope and has doubled in participation since its inception in 1989. Past conferences have been held in France, Germany, Japan, and the United States.

In order to keep track of current global developments in a fast growing field, the Parallel CFD Conference annually brings individuals together to discuss and report results on the utilization of parallel computing as a practical computational tool for solving complex fluid dynamic problems. The purpose of this volume is to disseminate results of research conducted during the past year. The content of the papers included in this volume suggest that there is considerable effort being made to utilize parallel computing to solve a variety of fluid dynamics problems in topics such as climate modeling, consultation, aerodynamics, and many others.

The papers presented at the 1995 conference included subject areas such as novel parallel algorithms, parallel Euler and Navier-Stokes solvers, parallel Direct Simulation Monte Carlo method, parallel multigrid techniques, parallel flow visualization and grid generation, and parallel adaptive and irregular solvers. The applications of interest included the following: reacting flows, rarefied gas flows, multiphase flows, and turbulence; vehicle design, hypersonic reentry problems, and aerodynamic flows; challenges such as moving boundaries, interfaces, free surfaces and fluid-structure interactions; and parallel computing in aeronautics, astronautics, mechnical engineering, and environmental engineering.

We would like to thank all of the sponsors of the 1995 conference whose support was vital to the success of this conference. Special recognition should be given to CRAY, IBM, Intel, NASA-Ames Research Center, and Silicon Graphics for their financial support. In addition, we would also like to recognize and thank the organizing committee who make the annual international conference of Parallel CFD possible.

The Editors

ACKNOWLEDGEMENTS

Sponsors of the Parallel CFD'95 Conference

CRAY
IBM
Intel
NASA-Ames Research Center
Silicon Graphics

Conference Organizing Committee

S. Taylor - Chairperson, California Institute of Technology, USA
R. Agarwal, Wichita State University, USA
A. Ecer, Purdue University -Indianapolis (IUPUI), USA
D. Emerson, DRAL, UK
I. Foster, Argonne National Laboratory, USA
P. Fox, Purdue University -Indianapolis (IUPUI), USA
A. Geiger, RUS Stuttgart, Germany
J. Haeuser, Center for Logistic and Expert Systems, Germany
D. Keyes, Old Dominion University and CASE, USA
C. Lin, National Tsing Hua University, Taiwan
S. Peigin, Tomsk University, Russia
R. Pelz, Rutgers University, USA
J. Periaux, Dassualt Aviation, France
D. Roose, Katholike Universiteit, Belgium
N. Satofuka, Kyoto Institute of Technology, Japan
P. Schiano, CIRA, Italy
H. Simon, Silicon Graphics, USA
M. Vogels, NLR, The Netherlands
D. Weaver, Phillips Laboratory, USA

TABLE OF CONTENTS

5. Spectral Methods

6. Large Scale Applications

7. Performance Issues

8. Flow Visualization

9. Multigrid Methods

10. Applications

12. Adaptive Schemes

13. Climate Modeling

14. Navier-Stokes Solvers

15. Distributing Computing

16. Mesh Partitioning

17. Internal Flows

18. Software Tools

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Multiblock Structured Algorithms in Parallel CFD

C. de Nicola[a], R. Tognaccini[a] and P. Visingardi [b]

[a]Dipartimento di Progettazione Aeronautica, University of Naples,
P.le Tecchio 80, 80125 Naples, Italy

[b]IRSIP, Parallel Informatic Systems Research Institute, CNR,
via P. Castellino 111, 80131 Naples, Italy

1. INTRODUCTION

Realistic flow field simulations in aeronautical applications need large computer memory and CPU time. In spite of the significant progress of recent algorithms in accuracy and efficiency, the analysis of complex flow fields is still an expensive task; the advent of parallel machines seems to be the solution to drastically reduce the computation time and to permit a more extensive application of CFD methods.

The aim of the present work is to analyze some aspects of a widely diffused solution strategy, the Multiblock Structured Algorithm (MSA) and in particular to discuss on its peculiarities when applied in a parallel environment. From an aerodynamicist viewpoint, some of the main problems of MSA parallel implementation will be underlined. It will be highlighted, in particular, how aerodynamic requirements can often be in conflict with choices ruled by optimum parallel performance. Furthermore, stability considerations and their implication on the parallel performance will give the opportunity to indicate the main guidelines to follow in the application of these methods.

The MSAs were firstly addressed to overcome problems related to practical calculations by CFD methods. As far as we know the first application was performed at the Boeing CFD Laboratories [1]; the simulation of the Euler flow around a propfan/nacelle/wing configuration was limited by the central memory available at that time. The problem was overcome by loading in memory each block at time and performing one time step of a Runge-Kutta scheme. Later on this technique was more precisely defined in order to solve the problem of the grid generation around very complex aircraft configurations [2–5]: until this time, in fact, the high improvement in flow simulation algorithms and computer performance had not been supported by efficient grid generation tools. By dividing the domain into geometrically simpler subdomains (blocks), it was possible to realize computational grids around arbitrary domains.

It is here made a major distinction between what we call Domain Decomposition Technique (DDT) and the MSA. We refer to DDT when the parallelization is obtained by a simple data re-arrangement without affecting the numerical algorithm. The more general multiblock approach consists of dividing the differential problem into a number of parallel mixed initial boundary value problems exchanging data via artificial interface boundary

conditions:

$$\frac{\partial W_b(r_b,t)}{\partial t} + AW_b(r_b,t) = 0 \qquad b = 1,..,nb \tag{1}$$

$$\begin{array}{ll} I.C.: & W_b(r_b,0) = W_b^0(r_b) \quad r_b \in \Omega_b \\ B.C.: & W_b(r_b,t) = g_b(r_b) \quad\;\; r_b \in \partial\Omega_b \end{array} \tag{2}$$

$$\Omega_1 \cup \ldots \cup \Omega_b \cup \ldots \cup \Omega_{nb} = \Omega$$

Since additional internal boundary conditions are introduced at block interfaces, the original single domain scheme can be altered; in particular accuracy and stability can be affected. Whereas in literature several works on the accuracy of Multiblock approaches are available [6–8], the problem of their stability has been poorly discussed, although we recognized, as will be shown in the follows, their impact on computational performance.

An example of the capabilities of present MSAs is given in Figure 1, where the Euler pressure distribution around a Supersonic Commercial Transport configuration designed by Alenia (the main Italian aerospace industry) is presented. The solution was obtained by applying the Euler/Navier-Stokes flow simulation system developed at CIRA in cooperation with the University of Naples [9,10]. Interesting results concerning the parallel computations on the same configuration were presented by CIRA specialists [11].

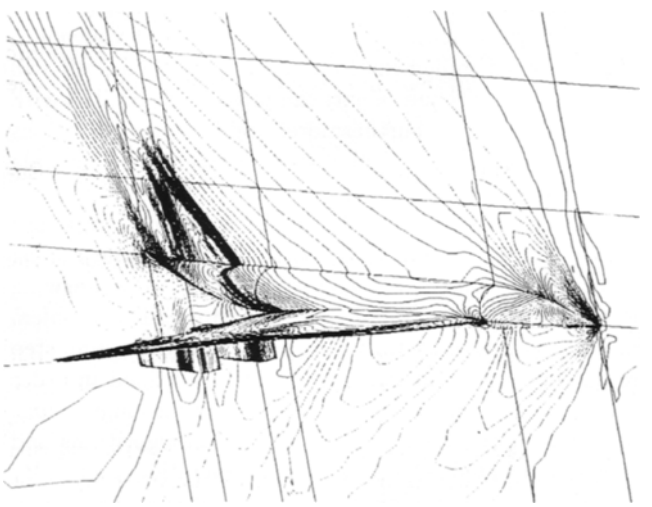


Figure 1. Euler flow solution around Alenia SCT3 configuration.
$M_\infty = 2$, $\alpha = 4^0$, isobars ($\Delta Cp = 0.005$).

Although the advantages offered by the MSAs are sufficient to ensure their success, there are a number of drawbacks. First of all, it is necessary the introduction of another

task in the flow simulation process, the block decomposition, that is difficult to automate and nowadays still requests a high percentage of the calculation turn-around time. Furthermore additional artificial boundary conditions are required by the introduction of block interfaces: to really improve grid generation flexibility, discontinuous grid lines across blocks should be allowed while preserving solution accuracy and stability. Anyway the most important point in this context is related to their parallel implementation. Are the MSAs intrinsically parallel? That is, do the efficiency and the speed-up reach satisfactory values in practical applications?

2. PARALLEL PERFORMANCE

Even if MSAs are widely used on parallel machines they do not straightforward ensure high performance. The well known performance indices of parallel systems are the speed-up (S) and the efficiency (ε), that respectively measure the earnings and the effective cost of the computation. These indices are generally far from ideal values for several causes; in particular we here deal with the load overhead, that is present when the computational load is not uniformly distributed among the processors, and with the communication overhead, that is caused by the physical swapping of data and information among processors and by the waiting time among the communications.

2.1. Load overhead

The load overhead problem is critical due to several conflicts between aerodynamic and parallel constraints:

- the grid is properly refined in each block on physical basis following aerodynamic requirements; a load balancing control at this time would further complicate the grid generation task;
- different flow models imply different type of loads in different regions (a zonal Euler/Navier Stokes model is a clear example);
- the computational load can dynamically change; see for instance the problem of the Aerodynamics of reacting flows.

The load balancing is the real problem of MSAs; it consists in minimizing the idle time related with communications among blocks. A way to perform it is to use a different block decomposition during the grid generation task and during the parallel computation since a topology decomposition only addressed to balance the load can be more easily automated. Anyway grid quality constraints across blocks could be more stringent reducing grid generation flexibility. This is the reason why this kind of technique provided the best results in conjunction with DDT. An example is given by the masked multiblock proposed by CIRA [12].

For steady state applications another possibility could be to asynchronize the communications among blocks, that would be equivalent, on a block scale, to use a local cell time step. Anyway, at present, there is no theoretical guaranty of stability and convergence of such an algorithm.

2.2. Communication overhead

The communication overhead is another main problem. In the near future it is expected that its influence on parallel performance will be reduced since it is strongly related to the available hardware technology. An evaluation of the communication overhead is provided by the fractional communication overhead (F_c) defined as the ratio between the communications and the computations, that is, in our case, the ratio between the block surface and the block volume in the computational domain. In practice F_c can be very high providing small efficiencies. The trivial reduction of F_c, obtained by increasing "ad hoc" the number of grid points per block, is not the proper solution, since the correct grid sizing depends on the aerodynamic accuracy. When massively parallel MIMD architectures are used, the F_c factor becomes critical and can only be reduced by diminishing communications. This can be done, for instance, by modifying internal boundary conditions; in this case, since the numerical algorithm results different from the single block case, the stability properties can be again affected.

3. STABILITY

The introduction of internal boundary conditions, as well as the strategies proposed to reduce the overheads, suggested to examine the consequences on the numerical stability. In particular we studied the problem with reference to explicit Runge-Kutta like schemes widely used for the numerical solution of initial value problems [13]. The question is: if we minimize communications, for example by freezing boundary conditions updating once per time step instead of once per stage, do we reduce the stability range and/or the convergence speed of the scheme?

Firstly it has been investigated the practical occurrence of instabilities due to artificial internal boundary conditions. The subsonic flow field solutions around the same 2D grid (NACA 0012 airfoil) respectively divided in 1 and 4 blocks are shown in Figures 2a,b. A standard 4-stage scheme with the stability limit CFL (2.8) was used for both the cases without smoothers (enthalpy damping, residual averaging). In the second test the internal boundary conditions were updated only once per time step: the unstable modes arising at block interfaces are clear. Only by halving the CFL number it was possible to get convergence.

The stability analysis was performed for 1D model problems by means of two classical eigenvalues methods: the Spectral Analysis of the Transitional Operator [14] and the more flexible Normal Mode Analysis [15]. An example of the obtained results relative to the advection-diffusion equation is shown in Figure 3. The used scheme was a two-stage Runge-Kutta with central space discretization; in the picture the non dimensional time step limit (ρ_{limit}) for the 1 block and the 2 block cases are plotted versus the Runge-Kutta coefficient of the 1^{st} stage (α_1). Unfortunately stability limits of the multiblock scheme revealed much lower at α_1 values providing the largest time steps for the one-block scheme. These results were in perfect agreement with the numerical experiments. A simple remedy that has been theoretically proved for the model problems and extended with success to practical Euler calculations consisted in reducing time step only near interfaces. In Figure 4 the convergence histories obtained for the airfoil test with and without the Local Time Step (LTS) correction are compared with the reference one-block case showing that the

original one-block convergence history has been practically recovered. The application of the LTS correction to parallel computing provided particularly interesting results: in fact communications could be strongly reduced preserving the convergence speed of the one-block scheme. The performance (S, ε) of quasi-1D Euler nozzle flow computations are summarized in table 1.

Table 1
Parallel Performance

		Number of nodes →					
		2		4		8	
		S	ε	S	ε	S	ε
	4	1.30	0.65	1.32	0.33	1.58	0.20
	1	1.71	0.85	2.63	0.66	3.80	0.47
	1/2	1.91	0.95	3.16	0.79	4.92	0.61
Upd/it	1/4	1.95	0.97	3.45	0.86	5.69	0.71
	1/6	1.99	0.99	3.66	0.91	6.12	0.76
	1/8	2.00	1.00	3.66	0.91	6.31	0.79
	1/10	2.00	1.00	3.69	0.92	6.44	0.80

Quasi-1D Euler nozzle flow: CFL=2.8, 4 stages Runge-Kutta scheme

They have been obtained by using the CONVEX MPP Metaseries of CIRA and PVM software as message passing tool. The total number of grid cells was fixed, the loads were perfectly balanced with the number of blocks equal to the number of nodes so that only the communication overhead was treated. The first and the last row values were respectively obtained by updating the internal boundary conditions at each stage of the Runge-Kutta scheme (equivalent to a DDT) and every 10 time steps (1/10 Upd/it) with a global communication reduction equal to 40. The improvement in parallel performance was obvious.

Similar stability problems are also present in implicit MSA as discussed in [16]. In particular, the convergence rate strongly decays by increasing the number of blocks when explicit internal boundary conditions are used (the simplest and most efficient procedure). Recent progress obtained by using a sort of coarse grid correction are reported in [17] and show that also for implicit MSA an accurate tayloring of the method can provide significant improvements in computational performance.

4. CONCLUDING REMARKS

The aim of this work was to underline some typical problems related to MSA and, in particular, to their application in parallel CFD. Beyond the indisputable features of the method, there are a number of points to always keep in mind when MSA are going to be applied.

The first one regards the clear conflict between the aerodynamic and the parallel requirements limiting the computational efficiency or the functionalities of the method.

The second argument of our discussion concerned with the relevance of the stability

analysis of MSA being their convergence rate strongly reduced by interface boundary conditions for both explicit and implicit schemes. We have suggested the LTS correction to reduce or eliminate these problems in case of multistage explicit methods enabling moreover a significant communication overhead reduction.

This is a first step towards an asynchronous communication approach also solving the load overhead problem for steady state calculations without the use of load balancing pre-processors. An interesting convergence acceleration technique, based on similar considerations, has been presented in [18]. It consists of processing only blocks with larger residuals, therefore reducing the total computational work. Anyway only supersonic applications were presented, while the more critical elliptic problems should be studied in detail to assert the real capability of this convergence accelerator.

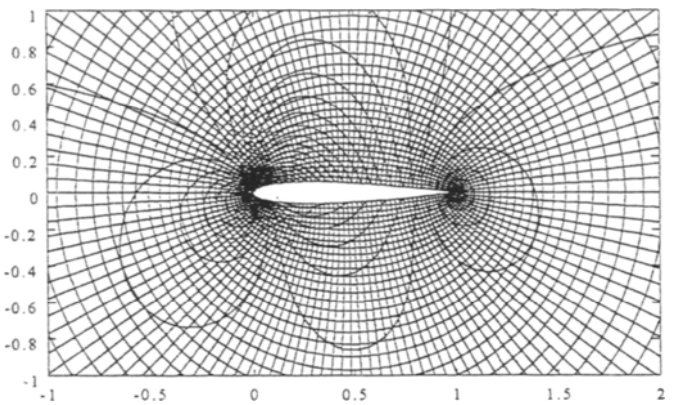


Figure 2a. NACA 0012 airfoil. 1 block. $M_\infty = 0.63$, $\alpha = 2^0$, $CFL = 2.8$.

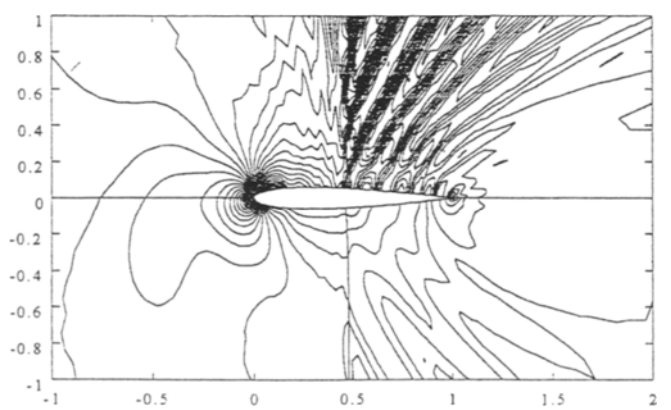


Figure 2b. NACA 0012 airfoil. 4 blocks. $M_\infty = 0.63$, $\alpha = 2^0$, $CFL = 2.8$ (500 iterations).

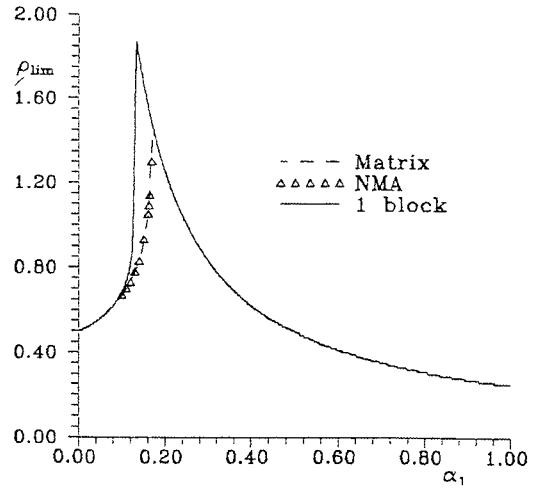


Figure 3. Two-stage stability limits for the advection-diffusion equation
Matrix: Spectral Analysis of the Transitional Operator.
NMA: Normal Mode Analysis.

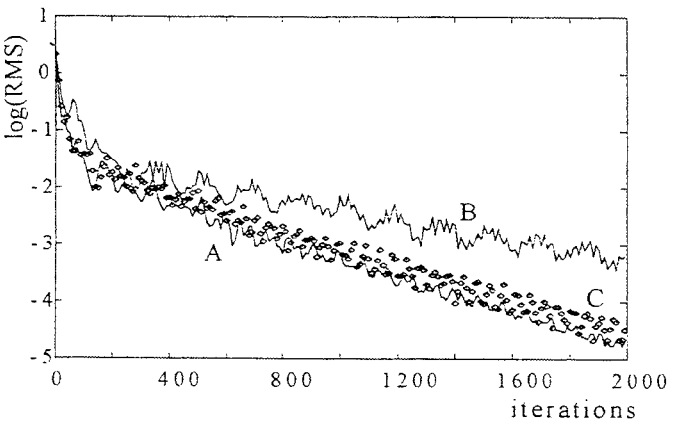


Figure 4. Euler flow convergence history. $M_\infty = 0.63$ $\alpha = 2^0$
A: 1 block, CFL=2.8; B: 4 blocks, CFL=1.4; C: 4 blocks, CFL=2.8 LTS correction.

REFERENCES

1. N. J. Yu, S. S. Samanth, P. E. Rubbert, Flow Prediction for Propfan Configurations Using Euler Equations, AIAA Paper 84-1645, 1984.
2. N. P. Weatherill, J. A. Shaw, C. R. Forsey, K. E. Rose, A Discussion on a Mesh Generation Technique Applicable to Complex Geometries, AGARD Symp. on Applications

of CFD in Aeronautics, Aix-En-Provence, France, 1986.
3. A. Amendola, R. Tognaccini, J. W. Boerstoel, A. Kassies, Validation of a Multiblock Euler Flow Solver with Propeller-Slipstream Flows, AGARD CP 437 Vol. 2, 1988.
4. J. W. Boerstoel, J. M. J. W. Jacobs, A. Kassies, A. Amendola, R. Tognaccini, P. L. Vitagliano, Design and Testing of a Multiblock Grid-Generation Procedure for Aircraft Design and Research, 64^{th} AGARD Fluid Dynamics Panel Meeting, Loen, 1989.
5. H. C. Chen, N. J. Yu, Development of a General Multiblock Flow Solver for Complex Configurations, 8^{th} GAMM Conference on Numerical Methods in Fluid Mechanics, Delft, 1989.
6. L. E. Eriksson, Numerical Solution of the Euler Equations Using Local Grid Refinement, FFA Report, 1988.
7. K. A. Hessenius, M. M. Rai, Three-Dimensional, Conservative, Euler Computations Using Patched Grid Systems and Explicit Methods, AIAA Paper 86-1081, 1986.
8. A. Kassies, R. Tognaccini, Boundary Conditions for Euler Equations at Internal Block Faces of Multiblock Domains Using Local Grid Refinement, AIAA Paper 90-1590, 1990.
9. C. de Nicola, R. Tognaccini, P. Visingardi, L. Paparone, Progress in the Aerodynamic Analysis of Inviscid Supersonic Flow Fields around Complex Aircraft Configurations, AIAA Paper 94-1821, 1994.
10. C. de Nicola, R. Tognaccini, A. Amendola, L. Paparone, P. L. Vitagliano, Euler Flow Analysis of Supersonic Transport Aircraft Configurations for Aerodynamic Optimization in Transonic and Supersonic Regimes, 19^{th} ICAS Congress, Anaheim, USA, 1994.
11. M. Amato, A. Matrone, L. Paparone, P. Schiano, A Parallel Multiblock Euler/Thin Layer Navier Stokes Flow Solver, Parallel CFD'94 Conference, Kyoto, Japan, 1994.
12. S. Borrelli, A. Matrone, P. Schiano, A Multiblock Hypersonic Flow Solver for Massively Parallel Computer, Proc. of Parallel CFD'92, May 92, North-Holland, 1993.
13. Van der Houwen, Construction of Integration Formulas for Initial Value Problems, North Holland, 1977.
14. C. de Nicola, G. Pinto, R. Tognaccini, On the Numerical Stability of Block Structured Algorithms with Applications to 1-D advection-diffusion problems, Computers&Fluids, Vol.24, N.1, pp 41-54, 1995.
15. C. de Nicola, G. Pinto, R. Tognaccini, A Normal Mode Stability Analysis of Multiblock Algorithms for the Solution of Fluid-Dynamics Equations, to appear on J. of Applied Numerical Mathematics.
16. C. B. Jenssen, Implicit Multiblock Euler and Navier-Stokes Calculations, AIAA Journal, Vol. 32, N.9, September 1994.
17. C. B. Jenssen, P. A. Weinerfelt, A Coarse Grid Correction Scheme for Implicit Multi Block Euler Calculations, AIAA Paper 95-0225, 1995.
18. M. L. Sawley, J. K. Tegner, A Data- Parallel Approach to Multiblock Flow Computations, Int. J. for Numerical Methods in Fluids, Vol. 19, 707-721, 1994.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Using Hundreds of Workstations for Production Running of Parallel CFD Applications

Craig J. Fischberg*
Chae M. Rhie*
Robert M. Zacharias*
Peter C. Bradley*
Tom M. DesSureault†
Pratt & Whitney, 400 Main Street, East Hartford, Connecticut 06108

Abstract

This paper describes the many software and hardware systems that were developed at Pratt & Whitney (P&W) to harness the power of hundreds of networked workstations. Historically, expensive dedicated supercomputers were used to perform realistically sized CFD analyses fast enough to impact hardware designs. With the large scale introduction of workstations a new less expensive computing resource became available to replace the supercomputer for the large scale execution of production, design oriented, CFD analyses. The software created to facilitate all aspects of the parallel execution of a three-dimensional Navier-Stokes solver are described. The evolution of this software and the hardware required to make the system robust, reliable, efficient and user-friendly in a large scale environment are also presented. The impact this parallelization has had on the design process, product quality and costs are also summarized. It will be shown that the use of networked workstations for CFD applications is a viable, less expensive alternative to supercomputers.

Introduction

Through the late 1980's the only computer platform that provided the execution speed and memory required for solving the 3-d Navier-Stokes equations for turbomachinery was the vector "supercomputer". The large number of grid points required for accurate 3-d Navier-Stokes flow simulation creates the need for both high speed computers to handle the tremendous number of floating point operations in a reasonable time and large amounts of memory to hold the data associated with all the grid points. In the 1980's, only a supercomputer could come close to meeting both of these demands.

* CFD Group, MS 163-01, email: fischbcj@pweh.com, rhiecm@pweh.com, zacharr1@pweh.com, bradlepc@pweh.com

† Network Services Group, MS 101-01, email: dessurtm@pweh.com

In 1986, Pratt & Whitney (P&W) purchased a Cray XMP supercomputer with 8 Mwords (64 Mbytes) of memory (RAM) at a cost of $13M to meet the needs of its high intensity computing applications such as 3-d Navier-Stokes solvers. For the next two years, P&W's Navier-Stokes solver, NASTAR, as well as other computational fluid dynamics (CFD) and structures applications were executed and developed using this XMP. The limitations of this high priced machine quickly became apparent: as the demands for the XMP grew so did the time jobs spent in queues waiting to be executed, even though the machine had 64 Mbytes of memory (high for the time), it was not sufficient to run fine enough meshes for most design applications and taking full advantage of the machine's vector processor required specialized, machine specific programming. To run a compressor fan analysis with a coarse mesh of 100,000 grid points, NASTAR required 16 Mwords of memory (including solid state disk) and 2-3 days to converge (due to the amount of time spent waiting in a queue plus CPU time). To run the desired mesh size of 500,000 grid points would have required 80 Mwords of memory, more than was available on the Cray XMP and would take too much time to have an impact on the design process.

In 1989, P&W made the decision to purchase workstations to replace mainframes for design/drafting work. The mainframes were expensive, continuously overloaded and provided crude graphics capability for design work. The decision was made to purchase Sun Sparc2 workstations configured with 32 MB of RAM to be used for design/drafting and small scale scientific computing. Even though the Sparc2 was a viable tool for drafting purposes its computational power for Navier-Stokes applications was limited. Running NASTAR with the 100,000 point coarse grid for a fan required 64 MB of RAM and one week of *continuous* execution time to converge. Most of the workstations that were purchased only had 32 MB of memory and could not be dedicated solely to the execution of NASTAR. The bottom line was that running large cases on single workstations was possible but a step back in turnaround time. Furthermore, the size of the grids that could be used was severely limited. The cost, however, was dramatically reduced. Each workstation cost only $20,000.

In mid-1989, after suffering through the limitations imposed by the Cray XMP and seeing no improvement offered by the workstations a fan designer proposed the idea of subdividing a standard, monolithic geometric model of a fan into pieces that could each be analyzed separately on different workstations. By utilizing this domain decomposition technique the size of the grid could be expanded because now each *subdomain* was limited by the workstation's memory instead of the entire problem. The number of workstations used for a problem could be increased according to the total memory required. Furthermore, by subdividing the problem a speedup due to parallel processing could be realized because each subdomain was sized smaller than would have been used for the single, non-parallelized problem.

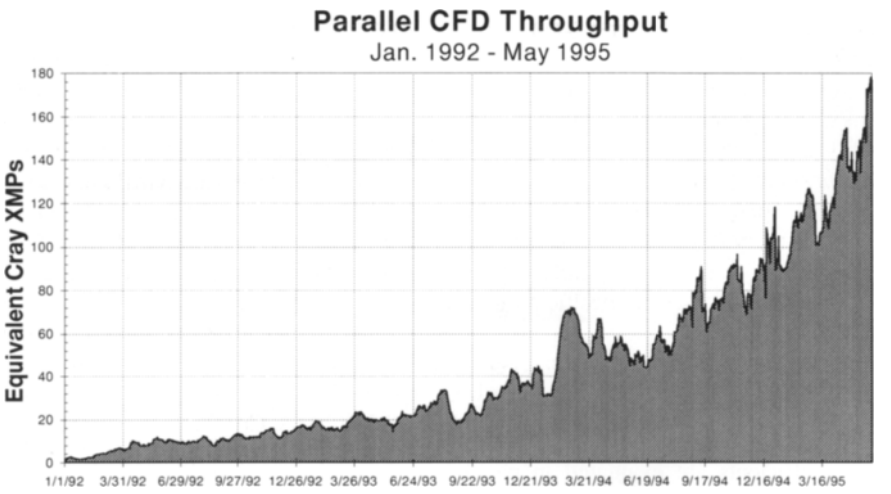


Figure 1. Daily parallel CFD throughput on Pratt & Whitney's Connecticut and Florida workstation networks.

By the end of 1989, the software to perform parallel CFD on networked workstations was in place. The innovative designer successfully executed a new fan design using this parallel computing environment at much less cost and time than was required for previous designs. As a result of the fan design success, this parallel CFD technology was quickly adopted by groups designing other engine components. Figure 1 shows the growth in computer usage for parallel CFD at P&W over time in terms of Cray XMP equivalents. The relative peaks correspond to large design efforts.

NASTAR: The Solver Parallelization

The first step of the parallelization process was to develop a scheme for distributing the work required to solve one single domain across multiple workstations. The algorithms in the three dimensional Navier-Stokes solver, NASTAR[1], were modified so that the original domain could be divided into *chunks* (sub-domains) such that each chunk could be solved as independently of the other chunks as possible.

In the solution procedure, the momentum equations are solved with a previously iterated pressure field. Since this preliminary velocity field does not satisfy the continuity equation, pressure correction equations are solved to establish new velocity and pressure fields which satisfy the continuity equation. A multi-step pressure correction procedure is used. One advantage of this pressure correction procedure is that only scalar matrices are solved for each equation. Coupling between the momentum and the continuity equations are achieved through the pressure correction procedure. The equations are solved in a sequential manner while each equation is solved in parallel. The momentum, energy, and turbulence scalar equations are solved with successive line under relaxation (SLUR) using the tri-diagonal matrix algorithm. The pressure correction equation is solved with a conjugate gradient matrix algorithm.

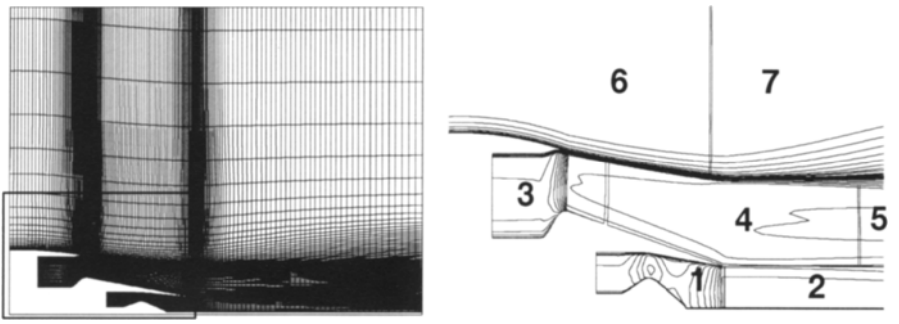


Figure 2: Jet engine nozzle grid and decomposition for parallel NASTAR.[2]

Domain decomposition is used for the parallel calculation. For example, Figure 2 shows an example of how the nozzle domain is broken up for a parallel calculation. The linearized influence coefficients are evaluated in parallel with explicit communication across the domain boundaries. Only the matrix solver is required to do more frequent or implicit communications. For the SLUR solver, each domain is solved independently and the domain boundary conditions are communicated a few times during the sweeping of the line inversions.

The solution stability depends on how complicated the flow field is and on how frequently the boundary conditions are communicated. Obviously, complex flow features such as flow separations or strong flow gradients across the domain boundaries can hamper convergence rate. To minimize the time to convergence a delicate balance between the convergence rate (number of iterations to convergence) and the communication overhead has to be found by numerical experiments.

In the present pressure correction procedure, the majority of the computational effort is spent in the pressure corrections. Therefore, it is essential to be very effective in solving the pressure correction equations. This efficiency is achieved by using a parallelized Preconditioned Conjugate Residual (PCR) solver, a conjugate gradient method. The parallel PCR solver is similar to a standard PCR algorithm[3] but interprocess communication has been added and some adaptations have been made to maximize parallelism.

The PCR solver uses two different types of communication. First, processes that share a domain boundary must swap boundary values at each PCR iteration using direct, point-to-point communication. Also, the solver's line searches require that a global inner product be calculated. In order to calculate this product, each process first calculates a local inner product and broadcasts it to the other processes. Broadcasted values are then combined by each process to form the global product.

In order to ensure convergence, it is necessary to precondition the original system $Ax = b$. A preconditioning matrix M should be chosen such that M approximates A and is easy to solve. Solution then proceeds on the preconditioned system $M^{-1}Ax = M^{-1}b$. For nonparallel solvers, an incomplete LU factorization of A is a good choice for M as it offers a good balance of convergence acceleration and CPU consumption. Unfortunately, inverting M

entails forward and backward sweeps that are inherently serial. The parallel PCR solver uses a modified LU factorization of *A* that allows each domain to do sweeps in parallel. The convergence rate is significantly lower than the rate for the unmodified factorization, but overall solver speed is much greater due to parallel execution.

Prowess: Communication System and Parallel Process Control

Several software systems were created or modified to run NASTAR in parallel. The first requirement was to create a system to facilitate the exchange of information between the solver processes running on different workstations. The Prowess (Parallel Running of Workstations Employing SocketS) communication system was designed to fulfill this need. Prowess was developed to isolate the solver developer from all the details of initiating multiple processes on different workstations, establishing communication channels between these processes and the low level mechanisms used to pass information. Initially, Prowess' sole role was as a communication facilitator but, as requirements for parallel job process control, accounting, reliability and workstation user protection were encountered, these too were developed into Prowess.

The number and complexity of the Prowess functions were minimized to simplify the already daunting task of parallelizing a complex CFD code consisting of over 100,000 lines of FORTRAN code. To achieve this simplification, the paradigm that was adopted for message passing was that of file I/O. The significance of this paradigm is that communication is performed in streams. Most message passing libraries such as PVM[4] and MPI[5] implement communication using discrete messages. Reading and writing data with Prowess, as when reading/writing files, if process A writes 20,000 bytes to process B, process B can read this data as four 5,000 byte messages, two 10,000 byte messages or one 20,000 byte message. Also, as with file I/O only two functions are required to read and write data, Slave_read() (islave_read() in FORTRAN) and Slave_write() (islave_write() in FORTRAN). By default, all communication uses blocking I/O so that neither function returns until the requested amount of data has been read or written. The Slave_read_select() (islave_read_slect()) function is available to determine which communication channels have data waiting to be read. Non-blocking communication can be implemented using this function.

Communication within Prowess is very efficient. To minimize message passing latency and maximize bandwidth all communication is performed directly between the communicating processes with no intermediary process touching the data. This technique is similar to running PVM in "route direct" mode, however, the deadlock problems that are common when using this mode of PVM have been eliminated in Prowess through the use of proprietary deadlock prevention algorithms. Figure 3 shows the performance of Prowess relative to PVM on different network types.

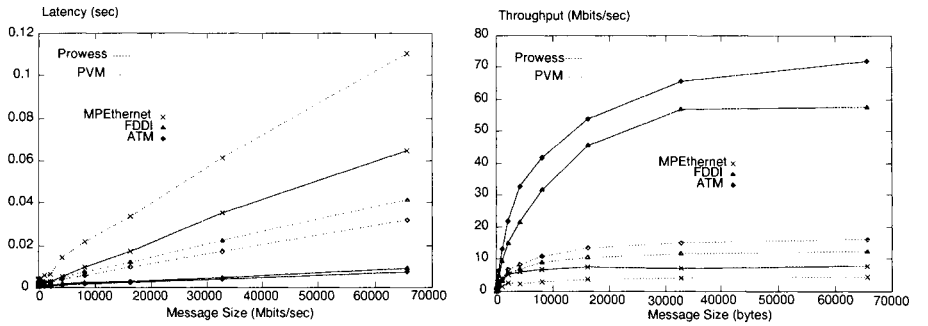


Figure 3: Prowess vs. PVM message passing latency and bandwidth.*

Once the foundation was laid upon which the solvers could be parallelized Prowess was developed to initiate the solver "slave" processes on each workstation and establish the appropriate communication channels. One of the command line arguments to Prowess is the name of an input file containing the names of the processes to run, the hosts on which to run them and interprocess connectivity information. When the "prowess" command is executed, a master process is initiated. This master process starts the slave processes provided in the input file on the hosts also provided in the input file and establishes communication channels as described in the input file connectivity table. After the slave processes are started, the master stays alive to monitor and control the slave processes until they have all terminated. The master also ensures that orphaned processes are terminated so that dead or runaway processes are not left alive all over the network.

To help the user control all the slave processes of a job, the master propagates the UNIX signals TSTP, CONT, INT, TERM, USR1 and USR2 that it receives to the slave processes. As a result, the user can terminate all slave processes at once by sending the appropriate signal to the master. At P&W, all parallelized applications have been written to checkpoint and terminate when they receive a UNIX TERM signal so that an entire parallel CFD job can be checkpointed and terminated by sending a TERM signal to the Prowess master.

Protecting the interactive workstation user was one of the major obstacles that was overcome to make parallel CFD on P&W's open network successful. The majority of workstations at P&W sit on the desktops of CAD users. These users perform mostly interactive work that can be severely impacted by the execution of CFD solvers on their workstations. Protecting these users from unwanted CFD processes was required in order to get permission to run parallel CFD on these workstations company wide. This protection is provided through Prowess in three ways: automatic morning shutdown, the "prowesskill" command and activity detection.

To prevent users from being impacted at login time on weekdays, Prowess is configured to automatically shutdown most parallel CFD jobs by 6:00 AM Monday through Friday. Both

* Measurements performed under the direction of Dr. Patrick Dowd, SUNY, Buffalo on NASA Lewis Lace Workstation Cluster

the time and days at which the shutdown occurs are configurable through the Prowess configuration file. The shutdown is performed by each master process by sending a TERM signal to its slaves. This action results in the solvers saving their current states and terminating. Each of the parallel jobs throughout P&W begins terminating at a random time between 5:30 AM and 6:00 AM to help reduce the peak network load that would occur if all jobs terminated simultaneously.

The prowesskill command was created to give workstation users absolute control over Prowess jobs running on any workstation. All users can use the prowesskill command to terminate *any* Prowess job executing on any workstation. The execution of prowesskill causes the parallel application to checkpoint and terminate. A job migration technique has also been implemented so that a prowesskilled application can continue executing on an unoccupied workstation.

P&W's policy is that the interactive user has first priority on a workstation. It is for this reason that the prowesskill command was created. Before its implementation, users had no guarantee that they could gain control of a workstation if a parallel application was running on it. Furthermore, many parallel CFD jobs were lost due to users rebooting workstations that were occupied by CFD slave processes. The prowesskill command provides users with a means to gain control of a workstation while preserving all the work performed by the parallel application up to the time of the prowesskill execution. With the implementation of process migration a prowesskilled application only suffers a minor delay.

The final user protection is provided through Prowess' activity detection options. Prowess' activity detection causes slave processes to be suspended or terminated as soon as any activity is detected on a workstation's keyboard or mouse. Either slave suspension or termination can be chosen by the Prowess user on a host by host basis. In most cases, users choose termination over suspension so that a job can be moved elsewhere to continue executing instead of remaining suspended for hours while a user continues to use a workstation. The termination occurs through the automatic execution of the prowesskill command on the workstation on which activity was detected.

Execution Techniques for Robustness

The initial attempts at parallel execution of NASTAR were less than 50% successful. Many factors combined to make parallel execution unreliable. Bugs in the workstation operating systems, volatility of file servers, network traffic spikes and other unpredictable events resulted in lost work and delays in turnaround time. Several techniques were developed to reduce both the vulnerability of NASTAR to these events and the amount of work lost if a catastrophic event does occur.

The most significant obstacles to reliability were presented by operating system and network interface bugs. Both Hewlett-Packard's HPUX 9.05 and Sun Microsystems' SunOS 4.1 contained bugs related to the acknowledgment of packets in the TCP protocol stack. These bugs caused the network between communicating slaves to be periodically flooded with tens of thousands of acknowledgment packets per second. Whenever one of these events occurred, the flooding of the network would continue until the parallel job

failed. Both of these bugs were fixed through patches provided by Hewlett Packard and Sun and do not exist in newer versions of their operating systems.

One bug created storms of address request packets (ARP's) at midnight causing communication failures. Others caused the corruption of large data files during the initiation and termination of NASTAR jobs and communication deadlocks. The reliability of parallel CFD jobs jumped from below 50% to over 90% following the elimination of these bugs. The major lesson learned through this experience is not to ignore the operating system as a possible cause for software system breakdowns.

Other events that continue to occur cause the failure of parallel CFD jobs. These events include slave workstation crashes and reboots, file server problems, network hardware failures, user error and application code bugs. Two approaches are used to reduce the impact of these failures. The first approach is to reduce the exposure of parallel jobs to these events and the second is to reduce the amount of work lost when one of these events occurs.

To reduce the exposure of parallel CFD jobs to network hardware failures the workstation selection algorithm limits the span of each job to a minimum number of network "segments." In this context a segment is any physical network wholly contained on one side of a router, bridge or switching hardware such as an ethernet segment or FDDI ring. The workstation search process attempts to find as many hosts as possible in one segment before searching in a new segment. This algorithm minimizes the number of points of failure in the network hardware to which jobs are exposed.

Initial runs of parallel NASTAR frequently failed due to poor reliability of the file servers on which the input and output data and executable files were stored. P&W uses Sun's Network File System (NFS) to provide a uniform file system image across all hosts. Parallel NASTAR initially utilized NFS to access both its input data and executable files. This technique was unreliable because NFS is not robust enough to handle the loads that were being placed on the file servers by hundreds of NASTAR process running simultaneously. NASTAR was vulnerable to this unreliability during its entire 13 hour execution period.

The reliance of parallel jobs on network file servers is now minimized by copying all input data and most executable code to the local hosts on which the solver slave processes are running. Further safety is provided by performing all file output to the local host during the solver execution instead of writing accross the network. The first steps taken by a NASTAR slave process are to create a directory in the /tmp directory and then copy its chunk of data and solver binary code into this new directory using the "rcp" command instead of relying on NFS. Typical slave processes require 7-12 Mbytes of input data and 8 Mbytes of executable files. By utilizing the local host for file I/O the window of vulnerability to file servers is reduced from 13 hours or more to the time required to perform the initial copies. Following the completion of the slave process, all output data is copied to the host from which the input data was initially copied again, using rcp.

Even after reducing NASTAR's vulnerability to file servers, catastrophic events still occur. The amount of work lost due to these events is reduced by providing a checkpointing capability in the NASTAR solver. Each solver process writes its current state to the local

disk at a user defined interval. A *checkpoint* process that is started by each NASTAR slave has the responsibility of detecting this output and copying it to the data repository. However, before copying the data, this checkpointer analyzes the solver output to make sure it contains a valid solution and that solution has not diverged. If the solution is invalid the checkpointer terminates the entire parallel job immediately so as not waste computer resources on a failing execution. Furthermore, by checking the output before copying, the history of the solution before failure is not overwritten by the latest invalid output.

Through the combination of the various techniques described above, parallel CFD at P&W has reached a reliability rate of 95-98%. As new causes for job failures are discovered both the Prowess and NASTAR software are modified to prevent these failures in the future. P&W's goal is to achieve 100% reliability for all parallel CFD jobs running on its production network regardless of the job size.

LSF™ and Wsfind: Finding Available Workstations

To bring the power of hundreds of workstations at P&W to bear on everyday design problems, the burden of finding idle computers for parallel CFD was removed from the user. P&W has thousands of UNIX workstations of varying power (including differing numbers of CPU's per host) and operating systems spread throughout its East Hartford, Connecticut site. The LSF™[6] (Load Sharing Facility) software from Platform Computing Corporation of Toronto, Canada is used to manage this large volume of resources. The wsfind software, written at P&W, utilizes LSF™ to locate idle workstations capable of executing NASTAR's slave processes.

The basis of LSF™ is a system for monitoring the load of every computer on the network and the resources available on those systems. Resources include available memory, swap space, space in /tmp, etc. whereas, system loads include paging rate, run queue lengths, percent CPU utilization, keyboard idle time, etc.[7] LSF™ gathers this information in 15 second intervals and stores it using a hierarchical "cluster" based system. Wsfind uses the application program interface (API) provided with LSF™ to access this load and resource information and make job placement decisions based on host loads and job requirements.

Over time P&W has developed a complex set of resource requirements that must be satisfied before a NASTAR job can begin execution. These requirements were designed to not only satisfy the physical requirements of a NASTAR slave process (e.g. memory) but also to protect interactive users and minimize the shared use of workstation CPU's to maximize performance. These requirements vary depending on the time of day of the search and the relationship of the job owner to the workstation owner. At no point in time is a NASTAR job allowed to start on a host with logged in users unless the host belongs to the group of the NASTAR job owner, has multiple CPU's or has had no interactive use for at least 60 minutes.

This wsfind/LSF™ system is used to schedule CFD jobs on hundreds of workstations daily at P&W's East Hartford, Connecticut, West Palm Beach, Florida and Canadian facilities. The lsbatch™ batch scheduling software that comes with LSF™ is being used at P&W Canada to perform batch scheduling of parallel CFD jobs across their network of 100+ IBM and Hewlett Packard workstations. Platform Computing Corp. is currently modifying

lsbatch™ to provide multi-cluster batch scheduling of parallel jobs across the thousands of hosts available at P&W in East Hartford and West Palm Beach. Prowess is closely coupled with LSF™ to take advantage of all existing and future capability provided by both LSF™ and lsbatch™.

Visgrid: Parallelized Grid Generation to Remove a New Bottleneck

Using hundreds of workstations to compute the complex aerodynamic flowfield through a modern turbofan engine allowed designers to quickly and confidently assess new aerodynamic design concepts on the computer instead of in a wind tunnel. As will be discussed in the last section, this ability has had a profound effect on the Pratt & Whitney product, because new design concepts can be evaluated much more quickly and cheaply than in the past. However, the speed at which a new design could be evaluated exposed a new bottleneck in the CFD design analysis procedure: the time required to create the computational model of the new design.

A typical compressor airfoil (rotor or stator) is modeled using 250,000 computational nodes, or *grid points*. This is an accuracy requirement that was established through extensive CFD code validation. The proper placement of the grid points is crucial to the accuracy of the simulation. The software that generated the mesh originally took up to 30 minutes to generate the mesh for a single airfoil row. A typical high pressure compressor contains 23 rows of airfoils, so generating the model for the entire compressor would take from up to 12 hours. This was not acceptable for design purposes, when many perturbations of a base design would have to be evaluated. To remove this grid generation bottleneck, the in-house grid generator Visgrid was parallelized using Prowess.

Visgrid is a quasi-three-dimensional mesh generator which is executed in two steps. In step one, the computational mesh is generated on axisymmetric surfaces that vary in the axial and circumferential directions. (Figure 4). In the original implementation of Visgrid, these surfaces were processed sequentially, beginning with the surface coincident with the inner diameter of the compressor and progressing with increasing diameter to the outer diameter of the compressor. In step two, the mesh is re-processed on radial surfaces that vary in the axial and radial directions (Figure 5). Again, these surfaces were processed sequentially in the original implementation of Visgrid.

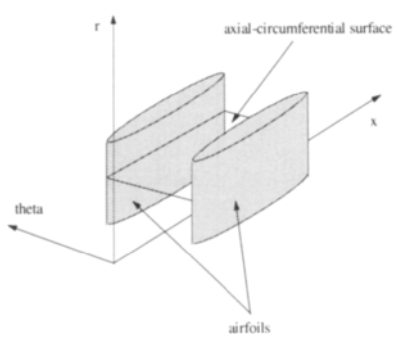


Figure 4. Axial-circumferential surface.

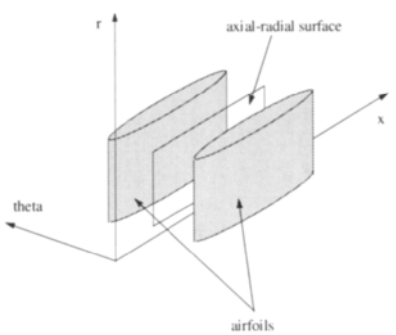


Figure 5. Axial-radial surface.

In order to speed the execution of Visgrid, Prowess was used to control the processing of these surfaces in parallel rather than in sequence. Prowess launches a *master* process that performs preliminary initialization. It also launches a number (typically 10) of *slave* processes that receive, process, and return a surface (either an axial-circumferential surface, step one, or an axial-radial surface, step two). The master process is like a card dealer, dealing cards to the players (slaves). The players play their card (process the surface) and return it to the dealer. The master receives the processed data, determines the next surface available for processing, and gives it to the slave. This technique of giving a surface to the next available slave "desensitizes" Visgrid to slow slaves. That is, a given slave can process at its own pace without slowing down its neighbors. This is possible because there is no interprocess communication (interdependency) between slaves, only between the slaves and the master. (Any interdependency is handled in the initialization step, and appropriate information is "funneled" to each slave.) This is a critical difference from the parallelization of the flow solver NASTAR, because in that application, there is a high degree of interdependency between slaves.

This parallelization of Visgrid reduced the grid generation time from up to 30 minutes per blade row to 5-8 minutes per blade row. This drastic reduction in grid generation setup time was a key ingredient in creating a CFD tool that allowed aerodynamic designers to quickly set up and numerically test new airfoil contouring concepts.

Computer Communication Network

The advent of parallel CFD on workstations resulted in a new emphasis on the Pratt & Whitney local area network (LAN). Running the parallel CFD jobs across the network resulted in a virtually continuous stream of ethernet traffic between all slave machines that increased proportionally with the number of workstations being utilized for all parallel jobs. This traffic was unlike the typical bursty LAN traffic and quickly resulted in ethernet utilizations of over 50%. This high utilization rate impacted interactive users connected to the same LAN and also the execution time of the CFD jobs. Thus, most Prowess executions were forced to run during off-shift periods (6 PM - 6 AM).

During the evolution of parallel CFD, efforts were continually focused on reducing the amount and impact of interprocess communications. Changes to both the LAN and the Prowess message passing software have played a complimentary role in the success of parallel computing. This work not only reduced the CFD execution times but also minimized the network impact on other applications and interactive users. This in turn, positively impacted the social and political effects of using idle desktop workstations normally used for CAD or other functions, for CFD analysis.

The network impact of the parallel workstation communications during parallel NASTAR execution has been studied and characterized over a number of years. The average effective network traffic rate for a typical single communications edge between two slave processes is currently 300-500 kbits per second. This average is calculated with a 1 second sample rate during a typical execution over many hours.

Workstation Type	Number per Ethernet Segment
Sun SparcStation 2 or less	12-24
Sun SparcStation 5	10
Sun SparcStation 10	8
Sun SparcStation 20 (2 CPU)	1

Table 1

With the increasing use of Prowess on idle desktop workstations, continuous monitoring of the LAN bandwidth utilization was implemented. Simple Network Management Protocol (SNMP) scripts were created polling all ethernets every 2 minutes for utilization and error information. This data quickly pointed out that network utilization was typically twice as high during "Prowess shifts" as during the normal business day. This forced new network design guidelines to be based on parallel CFD network requirements. These requirements resulted in a reduction of the number of workstations per ethernet segment. Today at Pratt & Whitney the number of prowess capable workstations per ethernet segment is shown in Table 1.

Pratt & Whitney Network Architecture

The overall network architecture is comprised of multiple 100 Mbits per second Fiber Distributed Data Interface (FDDI) backbone networks connecting approximately 200 ethernet segments as shown in Figure 6. The FDDI networks are connected with redundant Digital Equipment FDDI Gigaswitches. The ethernet segments are connected using multiport ethernet switching products that provide media rate bridging performance and a FDDI backbone network connection. The availability of low cost ethernet switching/bridging equipment has been a key enabler to the evolution of the Prowess parallel computing environment.

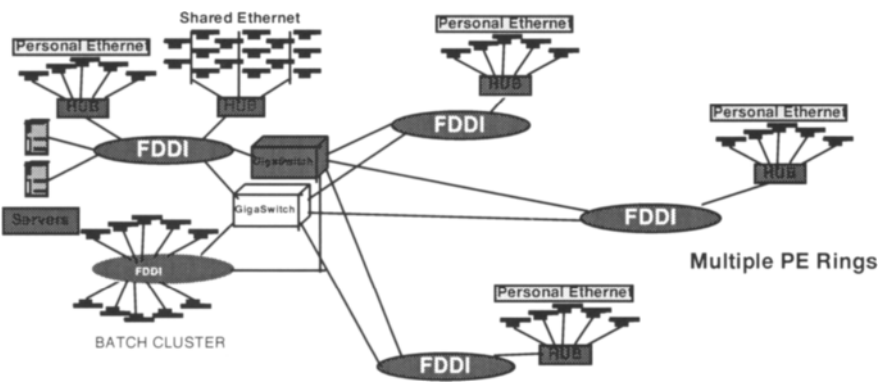


Figure 6. Pratt & Whitney, East Hartford network backbone.

Although there is a large number of ethernet segments, the network is still one logical Internet Protocol (IP) domain. The use of router centric network architectures (multiple IP domains or subnets) connecting large numbers of ethernets has been found to adversely affect prowess CFD application performance. This performance degradation is a function of the processing power of the routers with large numbers of ethernets, all heavily utilized for long periods. However, a combination of clusters of ethernet switches, connected by routers can also be effective with some additional administration to ensure most of the parallel CFD executions are contained on individual subnets.

Another factor that has had a major affect on network design is the use of multiple processor workstations. These workstations are configured such that an interactive CAD user can be using the power of one CPU while a Prowess application uses the remaining CPU power. This translates to 24 hour workstation availability for parallel CFD applications. Efforts are continuing to minimize any impact to the interactive user including providing a dedicated or personal ethernet connection to all high end multi-processor workstation as indicated in Table 1.

Business Impact: Better Products in Less Time

What has been described so far is of academic interest but most importantly it is positively impacting the cost and quality of actual compressor airfoil designs. Figure 1 shows the volume of parallel CFD work performed on networked workstations at P&W in terms of equivalent Cray XMP supercomputers throughout the history of Prowess. Today, 40-50 Cray XMP's would be required to provide the equivalent throughput at a cost of $500M-$600M (in 1986) whereas 95% of the workstations used for parallel CFD were purchased for interactive daytime use. The additional hardware costs for using these machines for parallel CFD is minimal. The shear capacity of this system has not only allowed many designers to run simultaneous design analyses but has also dramatically accelerated the development of NASTAR and other P&W CFD codes.

The combination of high design CFD throughput and accelerated code development resulted in two hardware designs with dramatically increased performance in reduced time. An efficiency gain of .6% in fan efficiency and 1.0% in low pressure compressor efficiency were realized in the latest fan redesign that utilized parallel NASTAR and Prowess[8]. Not only was the performance improved but the time for design to hardware was reduced from 18 months to 6 months. Approximately $20M were saved due to reduced rig testing and improved fuel burn for the customer. A similar result was achieved in the redesign of the PW4000 high pressure compressor for which the design time was cut in half, the development cost was reduced by $17M while the compressor efficiency was increased by 2%[9]. These achievements result in more competitive products through improved quality and shorter time to market.

References

[1] Rhie, C. M., "Pressure Based Navier-Stokes Solver Using the Multigrid Method," *AIAA Journal*, Volume 27, Number 8, pp. 1017-8, August, 1989.

[2] Malecki, R. E. and Lord, W. K., *A 2-D Duct/Nozzle Navier-Stokes Analysis System for Use by Designers,* AIAA 95-2624, 1995.

[3] Canuto, C., Hussaini, M., Quarteroni, A., Zang, T., *Spectral Methods in Fluid Dynamics,* Springer-Verlag, New York, 1988, p. 454.

[4] Geist, Al, Beguelin, Adam, Dongarra, Jack, Jiang, Weicheng, Manchek, Robert and Sunderam, Vaidy, "*PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing,* The MIT Press, Cambridge, Massachusetts, 1994.

[5] Message Passing Interface Forum (MPIF), *MPI: A Message-Passing Interface Standard,* University of Tennessee, Knoxville, Tennesee, 1994.

[6] Zhou, S., Wang, J., Zheng, X. and Delisle, P., "Utopia: A Load Sharing System for Large, Heterogeneous Distributed Computer Systems," *Software: Practice and Experience,* December, 1993.

[7] *LSF User's Guide,* Platform Computing Corporation, 1994.

[8] Rhie, C. M., Zacharias, R. M., Hobbs, D. E., Sarathy, K. P., Biederman, B. P., Lejambre, C. R., and Spear D. A., "Advanced Transonoic Fan Design Procedure Based On A Navier-Stokes Method," *ASME Journal of Turbomachinery*, Vol. 116, No. 2, April 1994, pp. 291-7.

[9] Lejambre, C. R., Zacharias, R. M., Biederman, B. P., Gleixner, A. J. and Yetka, C. J., "Development and Application of a Multistage Navier-Stokes Solver, Part II: Application to a High Pressure Compressor Design," presented at The 40th ASME Gas Turbine Conference, Houston, Texas, June 5-8, 1995.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel Computation of Frontal Processes

Marc Garbey and Damien Tromeur-Dervout [a] *

[a]L.A.N, URA 740, University Claude Bernard Lyon 1,
Bat101, 43 Bd du 11 Novembre 1918, 69622 Villeurbanne cedex, France

In this paper we study the computation of combustion fronts using MIMD architecture with up to 200 nodes. Our applications in gas combustion, solid combustion as well as frontal polymerization are characterized by stiff fronts that propagate with nonlinear dynamics. The understanding of the mechanism of instabilities is the goal of our direct numerical simulation. The multiple scale phenomena under consideration lead to very intense computation. This paper emphasizes the special use of domain decomposition and operator splitting combined with asymptotic analytical qualitative results to obtain more efficient and accurate solvers for such problems.

1. Introduction

We consider the two following models of combustion front:

- first, a classical thermo-diffusive model describing the combustion of a gas with a two-step chemical reaction [10][16] [18] [1]. Such a model describes the appearance of cellular flames and more complex pattern formation, [3] [4]. This model has been analyzed rather intensively, but few numerical result seems to be available.

- second, a model describing the reaction process of frontal polymerization. Frontal polymerization has been studied for several years in former USSR [22][17][20][21] to design new materials. New aspect of frontal polymerization are currently investigated to design new materials that cannot be produced by classical processes [2]. Our first step in the modelization is to couple a reaction diffusion system well known in solid combustion [6] to the Navier Stokes equations written in Boussinesq approximation. Our interest in this second model is to study the interaction between a mechanism of convective UN-stability similar to Rayleigh Bénard instability and a mechanism of thermal UN-stability well known in solid combustion [6] [13]. The direct numerical simulation [9] complements our analytical work in [7] [8].

We will refer to *Model A* and *Model B* respectively as the first and respectively second items described above.

*The authors want to thanks the Oak Ridge National Laboratory and especially J. Dongarra for the access to the iPSC/860 machines and the Bergen University parallel Laboratory and especially Tor Sorevich for the access to the Paragon machine . These parallel codes were elaborated on the Paragon of the *Centre Pour le Développement du Calcul Scientifique Parallèle of Lyon* sponsored by Région Rhône Alpes

The former problems are characterized by stiff fronts that propagate with nonlinear dynamics. The understanding of the mechanism of UN-stabilities is the goal of our direct numerical simulation. The multiple scale phenomena under consideration lead to very intense computation. We make a special use of domain decomposition and operator splitting combined with asymptotic analytical qualitative results to obtain more efficient and accurate solvers [5][19] adapted to the nature of the solution of such problems.

2. Numerical method

Both models we consider here present one or several thin layers, so the numericals methods to solve these two problems follow the same approach.

The main difficulty to compute the solutions of our combustions problems are the zone(s) of rapid variation of the temperature and concentration of the reactant(s). More precisely, we have one (or two) transition layer(s) that correspond to the sharp combustion front: the Arrhenius nonlinear term is of order one *only* in these layer(s); we use this criterion to numerically locate the layers. The dynamics of the combustion process is driven by these thin layers; consequently it is critical to compute them very accurately.

The computation to obtain a highly accurate representation of the solution in the Transition Layer (TL) is of leading importance because the UN-stability of the front starts in the TL and then smoothes out in the regular domain. Therefore we design our numerical method essentially to solve this difficulty.

Let us notice for *Model B*, that derivation of the corresponding interface model [9] shows that in the limit of infinite activation energy the concentration exhibits a jump at the front, the temperature is continuous, but has a jump of its first order normal derivative to the front, and the velocity up to the third order normal derivative to the front stays continuous. Therefore the main difficulty on the computation is driven by the combustion process itself and not the computation of the flow.

Domain decomposition technique for combustion problems was introduced in [12]. We use the adaptive multiple domain decomposition technique of [5], which is specific to singular perturbation problems.

Let us describe briefly the method. We consider first the example of a scalar equation:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial z^2} + F_\epsilon(u),\ u(-1) = 0,\ u(1) = 1, \tag{1}$$

with some appropriate initial condition. We suppose that u exhibits a TL of ϵ thickness located at $z = S(t) \in (-1, 1)$ and we suppose that S is such that $F_\epsilon(u(z,t))$ reaches its maximum at $z = S(t)$, with $F_\epsilon = O(1)$.

We recall the properties of pseudospectral approximations with Chebyshev polynomials on a *single domain* of computation denoted Ω and with no adaptivity. Let Ω be a compact subset of the interval $[-1, 1]$ and $u \in C^\infty(\Omega)$. Let further h be an affine function from $[-1, 1]$ into Ω. In the Chebyshev pseudospectral method, $u(z)$ is approximated as a finite sum of Chebyshev polynomials:

$$P_N u(z) = \sum_{j=0}^{N} a_j T_j(h^{-1}(z)),$$

where $T_j(y) = \cos(j \cos^{-1}(y))$. The coefficients a_j are obtained by collocating the solution at the points $(z_j)_{j=0...N}$ such that

$$h^{-1}(z_j) = \cos(\frac{\pi j}{N}),\ j = 0 \ldots N.$$

Let e_N be the error: $e_N = u - P_N u$. We put $\tilde{u}(h^{-1}(z)) = u(z)$. Then the following a priori estimate takes place:

$$\mid e_N \mid \leq C^t \frac{\parallel \tilde{u} \parallel_p}{N^{p-1}},$$

where $\mid \cdot \mid$ is the maximum norm, p is an integer and C^t is a real number depending on the choice of the Sobolev norm $\parallel \parallel_p$. Since p is not specified the error seems to be smaller than any inverse power of N. However, the constant C^t grows as p increases according to the smoothness of u. In practice for stiff problems, which we consider here, the pseudospectral method is error prone and subject to spurious oscillations. To avoid this difficulty we introduce an adaptive grid. It was shown in [3], [5] that an efficient and accurate way to solve a TL is to use two subdomains with an interface located at the front and a mapping that concentrates the collocation points at the end of the subdomains in the layer.

To be more specific, we introduce two one-parameter families of mappings. Let S_{num} be a numerical approximation of $S(t)$. First we decompose the interval $[-1, 1]$ into $[-1, S_{num}] \cup [S_{num}, 1]$ and introduce a one-parameter family of affine mappings that corresponds to the domain decomposition:

$$g_1 :\ [-1,1] \longrightarrow \Omega_1 = [-1, S_{num}]\ \ y \longrightarrow z = g_1(y, S_{num}) \quad \textit{and}$$
$$g_2 :\ [-1,1] \longrightarrow \Omega_2 = [S_{num},\ 1]\ \ y \longrightarrow z = g_2(y, S_{num})$$

Now the solution u of (1) restricted to one of these subdomains exhibits a Boundary Layer (BL) in the neighborhood of S_{num}. Therefore we use a second family of mapping of BL type:

$$f_i :\ [-1,1] \longrightarrow [-1,1]\ \ s \longrightarrow y = f_i(s, \alpha)$$

with

$$f_i(s, \alpha) = \pm \left[\frac{4}{\pi} tan^{-1}(\alpha\ \ tan(\frac{\pi}{4}(\pm s - 1))) + 1\right], i = 1, 2$$

Here α is a small free parameter that determines the concentration of the collocation points in the physical space. $i = 1$ and $+$ (respt. $i = 2$ and $-$) corresponds to a boundary layer at the right end of the interval (respt. the left end). Other mappings can also be considered.

In singular perturbations, one uses stretching variables of the form: $\xi = (z - S)/\epsilon$, where S is the location of the layer and ϵ is a measure of the stretching. It is easy to see that the parameter α in the non-linear mappings f_1 and f_2 plays a role similar to ϵ in the numerical method.

The unknown function u on the interval $[-1, 1]$ is approximated by a piecewise polynomial $P_{N,i}$, $i = 1, 2$ with the condition of continuity of its first derivative at the interface. In the numerical computation of the problem (1), we find the maximum of the function

$F_\epsilon(u(z,t))$ and adapt S_{num} in time, depending on the speed of propagation of the front. We have observed in [5] that the best choice for the parameter α is asymptotically $\alpha \approx \sqrt{\epsilon}$. One can optimize the choice of the stretching parameter α by means of a priori estimate as in [4], [5].

Let us consider a semi-implicit Euler scheme for the time marching:

$$\frac{u^{n+1} - u^n}{\Delta t} = D^2 u^{n+1} + F_\epsilon(u^n), \quad x \in \Omega_i, \; i = 1,2, \tag{2}$$

where Δt is a constant time step, and D is the operator of differentiation. We take the second order derivative term implicitly because it gives the main constraint on the time step. Since the nonlinear term is taken explicitly, u^{n+1} can be found as a solution of the linear system

$$\tilde{D} u^{n+1} = u^n + \Delta t \, F_\epsilon(u^n). \tag{3}$$

with a matrix $\tilde{D}$ invariant in time. Because of the domain decomposition $\tilde{D}$ has the following block structure:

$$\tilde{D} = \begin{pmatrix} A_1 & \hat{\beta} & \\ \tilde{\alpha}^t & \gamma & \tilde{\beta}^t \\ & \hat{\alpha} & A_2 \end{pmatrix}$$

where

- γ is a real number,
- A_1 and A_2 are $(N-1) \times (N-1)$ matrices.
- $\hat{\alpha}$, $\hat{\beta}$, $\tilde{\alpha}^t$, $\tilde{\beta}^t$ are vectors with $(N-1)$ components.

The row $(\tilde{\alpha}^t, \gamma, \tilde{\beta}^t)$ appears from the condition of continuity of the first derivative at the interface. The linear system (4) can eventually be solved with two parallel processes. The domain decomposition with two subdomains that we have presented here can be generalized to more subdomains to solve multiple front structures [5]. This domain decomposition is a **first level of rough parallelism** that is a priori not scalable because of its dependence only on the numbers of the layers.

To move toward a **second level of parallelism** that is using a fine grid level, we consider now the generalization of our previous example (1) to the case of two space dimensions:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial z^2} + \frac{\partial^2 u}{\partial x^2} + F_\epsilon(u), \quad u(-1,x) = 0, \; u(1,x) = 1, \; z \in (-1,1), \; x \in (0, 2\pi) \tag{4}$$

with some appropriate initial condition and the condition of periodicity of the function $u(z,x)$ in x. We assume now that the TL of the problem (5) depends on x weakly. This hypothesis simplifies significantly the technique of the domain decomposition method since we can consider strip subdomains. However orthogonal mapping techniques in two space dimensions [11] might be used to solve the problems when the front is smooth but not close to a straight line. In the x-direction, we use a central finite-difference scheme of order 6 for

D_x^2, the discrete approximation of $\frac{\partial^2}{\partial x^2}$, on a regular grid with a step $h = \frac{2\pi}{N_x - 1}$. In theory, for regular problems where $\epsilon = O_s(1)$, the numerical accuracy of our method should be limited to the sixth order by the finite-difference approximation in the x- direction. Since the front is assumed to be stiff in the z-direction but not stiff in the x-direction, we can keep the numerical error of the finite-difference approximation of $\frac{\partial^2}{\partial x^2}$ *smaller* than the numerical error of the approximation of $\frac{\partial^2}{\partial z^2}$ with the pseudospectral approximation [5], [3]. Therefore it is not necessary to use a better approximation of $\frac{\partial^2}{\partial x^2}$ such as for example the Fourier approximation. The finite difference approximation of order 6 of the term $\frac{\partial^2 u}{\partial x^2}$ is treated *explicitly.* We observe that the spectral radius of D_x^2, the discrete approximation of $\frac{\partial^2}{\partial x^2}$, is asymptotically smaller with the sixth order finite differences than with the Fourier approximation. So in our computations the time step constraint due to the explicit treatment of the second order derivative is of the same order as the time step restriction due to the explicit treatment of the source terms, and of the same order as the accuracy requirement connected with the physics of the process. The numerical algorithm can exploit the fine grid parallelism due to the explicit dependence on x because the computation of $D_x^2 u^n$ is fulfilled with an explicit formula which includes only local values of the function u^n. It is different for the "global" pseudospectral polynomial explicit approximation in the x-direction which is less appropriate for the parallel computing and/or for the use of the cash memory on the RISC architecture. Finally, we observe that the splitting of the operator in (5) replicate the procedure of the asymptotic expansion in the shock layer of an analytical study, since the dependence on y occurs as a second order expansion term. Generally speaking it is possible to derive new parallel algorithm that exploit systematically this asymptotic feature [14].

However for steady problems analogous to (5), i.e:

$$\frac{\partial^2 u}{\partial z^2} + \frac{\partial^2 u}{\partial x^2} = G_\epsilon(x,y), \quad u(-1,x) = 0,\ u(1,x) = 1,\ z \in (-1,1),\ x \in (0,2\pi) \tag{5}$$

one should use a different approach that is implicit with respect to both space variables.

We look for Θ as a discrete Fourier expansion:

$$\Theta(z,x) = \Sigma_{k=\frac{-N_x}{2},\frac{N_x}{2}-1} \hat{\Theta}_k(z)\, e^{ikx}.$$

The functions Θ_k can be found from the system of k independent ordinary differential equations:

$$\frac{\partial^2}{\partial z^2}\hat{\Theta}_k - k^2\,\hat{\Theta}_k = \hat{F}_k,\ k = \frac{-N_x}{2}, \frac{N_x}{2} - 1.$$

These equations are then solved with the one dimensional same domain decomposition method than described above. This algorithm has been applied to the stream function equation in model B and can be extended in a straightforward way to implicit the scheme for the two dimensional diffusion term in the unsteady equations for the other unknowns.

However the cost of this implicit algorithm is no longer linear with respect to N_x, due to the Fourier transform. Also the matrices must be transpose and the parallel implementation of the scheme should carefully overlap communication by computation using some appropriate partitioning of the datas.

A third level of parallelism is obtained for combustion problems that are modelized by *system* of PDE's. As a matter of example, let us consider model A. The combustion model follows the sequential reaction mechanism:

$$A \rightarrow \mu B \rightarrow C,$$

where μ is the stoichiometric coefficient for the first reaction. Reactant A is converted to an intermediate species B prior to being burned and converted to the final product C.

This highly simplified model have been used with considerable success in modeling the combustion of hydrocarbons [23]. Here, we consider the case of a cylindrical flame stabilized by a line source of fuel. The problem involves three variables— T, a scaled temperature, C_1, the concentration of reactant A, and C_2, the concentration of the intermediate species B, all functions of the polar coordinates r and ψ— and corresponds to a thermodiffusive model for a two-step Arrhenius reaction [1]. The equations satisfied by T, C_1 and C_2 are,

$$\frac{\partial T}{\partial r} = \Delta T - \frac{K}{r}\frac{\partial T}{\partial r} + \zeta Da_1 C_1 R_1(T) + (1-\zeta) Da_2 C_2 R_2(T) \quad (6)$$

$$\frac{\partial C_1}{\partial r} = \frac{1}{L_1}\Delta C_1 - \frac{K}{r}\frac{\partial C_1}{\partial r} - Da_1 C_1 R_1(T), \quad (7)$$

$$\frac{\partial C_2}{\partial r} = \frac{1}{L_2}\Delta C_2 - \frac{K}{r}\frac{\partial C_2}{\partial r} + Da_1 C_1 R_1(T) - Da_2 C_2 R_2(T) \quad (8)$$

where R_1 and R_2 are the nonlinear source terms:

$$R_1(T) = \exp\frac{\beta_1(T-T_0)}{1+\gamma_1(T-T_0))}, R_2(T) = \exp\frac{\beta_2(T-1)}{1+\gamma_2(T-1))},$$

Δ is the Laplacian in polar coordinate. The parameters are L_1 and L_2, the Lewis numbers; Da_1 and Da_2, the Daemker numbers; ζ the heat release of the first reaction; and K, a measure for the strength of the fuel injection; $T_0 \in [0,1]$ is a (scaled) temperature close to the temperature at which the first reaction occurs. β_i is proportional to the activation energy E_i of each chemical reaction. The coefficients γ_i are function of the temperature of the unburned fresh mixture, the final adiabatic burned temperature and T_0. The activation energy of each chemical reaction is large and consequently β_i; $i = 1,2$ as well as one of the Daemker number at least are large. The derivation of this model but for cartesian geometry has been done for example in [18] [1].

We observe that these three equations are *weakly coupled in space* because coupled through the Arrhenius source terms

$$Da_i C_i R_i(T),\ i = 1,2$$

that are exponentially small *except* in the combustion layers; so the **third level of parallelism** is to split the three equations into three parallel subsets of computation matched to three subsets of nodes. In addition, we can improve the parallel efficiency of the scheme by restricting us to the communications of the only non exponentially small values of the Arrhenius source terms.

Details of the analysis, numerical validation, and simulation results can be found in [15]; we will describe shortly in the next section the performance of our method.

3. Parallel implementation results

3.1. Result on thermo-diffusive model

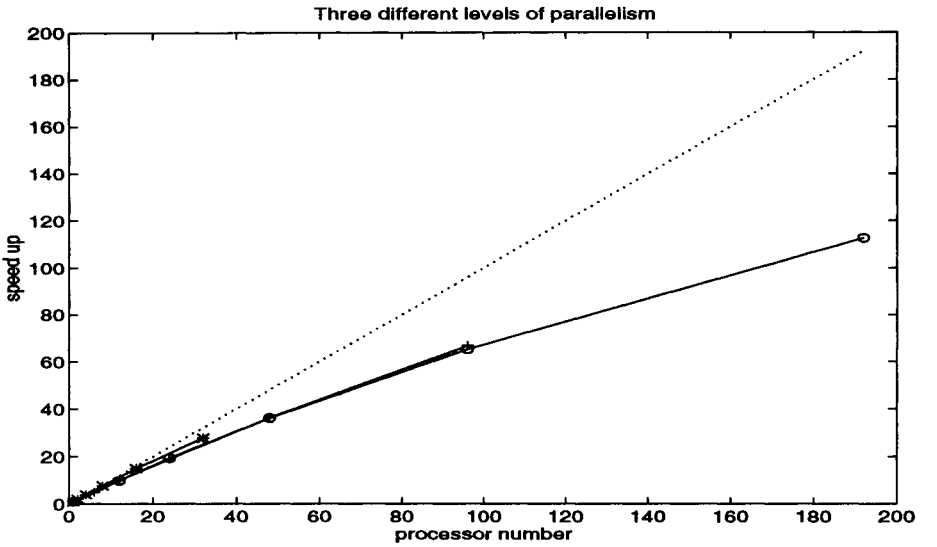


Figure 1.

Figure 1 shows, for the first *Model A* the effect on efficiency of the three following levels of parallelism:

- (a) splitting in the space variables, adapted to the frontal structure of our problems.
- (b) domain decomposition to solve a Transition Layer (TL) with the pseudospectral method.
- (c) splitting of the equations that are weakly coupled by the Arrhenius terms outside the TL.

For the first model with two space dimensional and a discretization with about 30.10^3 degrees of freedom, we obtain an efficiency of 87 per cent with 32 nodes using (a); using (a) and (c) we can run our code with 96 nodes and reach an efficiency of 70 per cent; using (a), (b) and (c), we can use up to 192 nodes and reach an efficiency of at least 60 per cent: for this last case, our code is running 145 times faster than the original optimized sequential code on a HP720 workstation and 15 times faster than a fully vectorized code running on a CRAY YMP that provides the same level of accuracy.

3.2. Result on the reaction process of frontal polymerization

We present here the result of strategy (a) and (b) applied to *Model B*. Let Nx be the number of points in the x direction where finite differences are used, nd the number of subdomain in the z direction where spectral discretization is performed, and N be the number of Tchebycheff point in each subdomain. In the different runs of Table 1, each processor processes one of the nd subdomains, except when only one processor is used. In this last case it performs the computation of the nd subdomains without any communications to itself. Table 1 shows an efficiency about 90% for the frontal polymerization

nb of proc / N_x	1	2	4	8	16	32	64
32	100%	95.9 %	91.6 %	83.7 %	-	-	-
64	100%	97.0 %	96.5 %	92.7 %	84.05 %	-	-
128	100%	97.9%	96.7%	94.6 %	92.1 %	82.7 %	-
256	100%	98.4%	98.3%	96.5 %	93.4%	82.2%	72.6 %

Table 1
Parallel Efficiency, $nd = 2$, $N = 59$

code. This efficiency seems to scale i.e when we double the number of Nx points and the number of processors, the efficiency stays relatively constant. This result seems to be not satisfied by the last result in the row Nx=256: however we have carefully checked that the memory swap explains this bad result.

We note that for a fixed number of processor the efficiency increases when Nx grows. So the non blocking messages strategy work well because a large part of the communication time comes from the start-up of the non blocking send which is costly than blocking send.

4. Conclusion

We have been able to use MIMD architecture with high efficiency and/or large number of nodes to compute non linear propagating phenomena such as combustion front. In our tow dimensional computation the size of the discretization problems were dictated by the accuracy requirement as well as the illconditionned nature of the operators: however we were able to use up to 192 nodes on a Paragon system for such small problems and reduce the computing time dramatically. Our technique relies on combining complementary levels of parallelism such as domain decomposition or operator splitting according to the asymptotic property of the mathematical models. We would like further to mention that our numerical simulations have been compared with great success against the linear and/or weakly non linear stability analysis for our two model problems.

REFERENCES

1. J. Pelaez, *Stability of Premixed Flames with two Thin Reactions Layers*, SIAM J. Appl. Math. **47** , pp. 781–799 (1987).

2. I. P. Nagy, L. Sike and J.A. Pojman, *Thermochromic Composite Prepared via a Propagating Polymerization Front*, J. of the American, Chemical Society, (1995),117.
3. A. Bayliss, M. Garbey and B.J. Matkowsky, *An adaptive Pseudo-Spectral Domain Decomposition Method for Combustion problem*, J. Comput. Phys. (to appear)
4. A. Bayliss, D. Gottlieb, B.J. Matkowsky, and M. Minkoff, *An adaptive Pseudo-Spectral Method for Reaction Diffusion Problems*, J. Comput. Phys. 81, pp 421-443 (1989)
5. M.Garbey, *Domain Decomposition to solve Transition Layers and Asymptotic*, SIAM J. Sci.Comput. Vol15 No4, pp866-891, 1994.
6. S. B. Margolis, H.G. Kaper, G.K. Leaf, and B.J. Matkowsky, *Bifurcation of pulsating and spinning reaction fronts in condensed two-phase combustion*, Comb. Sci. and Tech. **43**(1985) , pp. 127-165.
7. M. Garbey, A. Taik, and V. Volpert, *Influence of natural convection on stability of reaction fronts in liquid* Quart. Appl. Math (to appear).
8. M. Garbey, A. Taik, and V. Volpert, *Linear stability analysis of reaction fronts in liquid* Quart. Appl. Math (to appear).
9. M. Garbey and V. Volpert, *Asymptotic and Numerical Computation of Polymerization Fronts* submitted to J. Comp. Phys.
10. A.P. Aldushin and S.G. Kasparyan, *Doklady Akademii Nauk SSSR* (5) **244**, 67-70 (1979) (in Russian).
11. R. Duraiswami and A. Prosperetti, *J. Comput. Phys.* **98**, 254-268 (1992).
12. U.Ehrenstein, H.Guillard, and R. Peyret, *International Journal for Numerical Methods in Fluids*, **9**, 499-515 (1989).
13. M. Garbey, H.G. Kaper, G.K. Leaf, and B.J. Matkowsky, *Using Maple for the Analysis of Bifurcation Phenomena in Condensed Phase Surface Combustion J. of Symbolic Comp.* **12**, 89-113 (1991).
14. M. Garbey, H.G. Kaper, *Heterogeneous Domain Decomposition for Singularly Perturbed Elliptic Boundary Value Problems*, preprint Argonne Nat. Lab. submitted to SIAM. J. Num. Analysis.
15. M. Garbey, D. Tromeur-Dervout, *Parallel Computation of Frontal Polymerization to be submitted to JCP*
16. S.B. Margolis and B.J. Matkowsky, *Steady and Pulsating Modes of Sequential Flame Propagation* Comb. Sci. Tech., **27**, 193-213, (1982)
17. B.V. Novozhilov, *Proc. Academy Sci. USSR, Phys. Chem. Sect.* **141**, 836-838 (1961).
18. J.Pelaez, A Linan,*Structure and Stability of Flames with two sequential reactions*, SIAM J. Appli. Math 45 (4) 1985.
19. F.X Roux and D. Tromeur-Dervout, *Parallelization of a multigrid solver via a domain decomposition method*, Proceeding of the 7^{th} conference on Domain Decomposition Methods, Contemporary Mathematics Volume 180, p. 439-444, 1994
20. K.G. Shkadinskii, B.I. Haikin, and A.G. Merzhanov, *Combustion, Explosion, and Shock Waves*, **7**, 15-22, (1971) .
21. V.A. Volpert, V.A. Volpert, S.P. Davtyan, I.N. Megrabova, and N.F. Surkov, *SIAM J. Appl. Math.*, **52**, No. 2, pp. 368-383 (1992).
22. Ya.B. Zeldovich and D.A. Frank-Kamenetsky, Zh. Fiz. Khim., **12**, 100 (1938) (in Russian).
23. C.K. Westbrook and F.L. Dryer, Comb. Sci. Tech., **27**, 31-43, (1981).

This Page Intentionally Left Blank

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

Modelling the Global Ocean Circulation on the T3D

C. S. Gwilliam*

Southampton Oceanography Centre, Empress Dock, Southampton, SO14 3ZH, U.K.

Abstract

The Ocean Circulation and Climate Advanced Modelling Project is developing a global ocean model suitable for climate studies, especially the investigation of global ocean characteristics. The model will also act as a test bed for improved schemes which model ocean physics. The large memory and time requirements for such a study has led to the use of array processors. In this paper an outline of the model and the parallel code is given, along with some initial results. A fuller description of the model can be found in ([7],[13]).

1 Introduction

The OCCAM model is an array processor version of the GFDL Modular Ocean Model ([10], [12]). It is based on the Bryan-Cox-Semtner ocean general circulation model ([1],[3],[11]) which uses the time-dependent equations:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}.\nabla)\mathbf{u} + w\frac{\partial \mathbf{u}}{\partial z} + f \times \mathbf{u} = -\frac{1}{\rho_0}\nabla p + \mathbf{D}_u + \mathbf{F}_u, \tag{1a}$$

$$\frac{\partial S}{\partial t} + (\mathbf{u}.\nabla)S + w\frac{\partial S}{\partial z} = \mathbf{D}_S + \mathbf{F}_S, \tag{1b}$$

$$\frac{\partial T}{\partial t} + (\mathbf{u}.\nabla)T + w\frac{\partial T}{\partial z} = \mathbf{D}_T + \mathbf{F}_T, \tag{1c}$$

$$\rho = \rho(T, S, p), \tag{1d}$$

$$\rho g = -\frac{\partial p}{\partial z}, \tag{1e}$$

$$\nabla.\mathbf{u} + \frac{\partial w}{\partial z} = 0. \tag{1f}$$

Equations (1a)-(1c) describe the horizontal momentum, temperature and salinity changes, equations (1d)-(1f) describe the density, pressure gradient and incompressibility. The variables are $\mathbf{u}$, the horizontal velocity; w, the vertical velocity; S, the salinity; and T,

*Supported by a Cray Research Fellowship

the potential temperature. The pressure variable is p, the density is ρ. The Coriolis term, f, is

$$f = 2\Omega \sin(\theta),$$

where Ω is the earth's rotation rate and θ is the latitude. **D** represents diffusion and **F** the forcing terms.

The world is mapped on to a 3-D longitude-latitude-depth (i, j, k) grid. Equations (1) are discretised for this grid using a central finite difference scheme and are stepped forward in time with an explicit, leapfrog method. Unfortunately, there are very fast surface gravity waves in the ocean that restrict the timestep and increase the computational cost. To overcome this, the surface gravity wave part of the solution (the barotropic mode) is treated separately from the rest of the solution (the baroclinic mode), allowing a longer timestep to be used for the latter. A free surface scheme, in which the resulting tidal equations are timestepped explicitly, is used for the barotropic mode. Many timesteps are used to step the surface gravity wave field for each timestep of the baroclinic part of the model.

Ocean models need to represent the main ocean features (e.g. the Gulf Stream, eddies) which are O(10km). However, ocean basins are O(1000km) and, therefore, a large number of grid points is needed. The requirement for a fine grid resolution means that the baroclinic timestep must be reduced if numerical stability is to be retained. In practice, the baroclinic timestep is less than an hour for resolutions finer than 1°.

The ocean circulation changes and develops over many years and the model has to be run for a very large number of time steps. To run a high resolution, global model, a fast computer with a large amount of memory is required. In the past, vector processors went some way to meeting these requirements and the new array processors appear to meet them.

In Section 2 issues involved in the parallel model are described. The initial three year spin up is described in Section 3, including a description of some problems arising during the run, and future work is described in Section 4.

2 Main Features of the Model

The OCCAM code is a global circulation model. To avoid problems with the convergence of the longitude-latitude grid at the North Pole a two-grid method ([4],[5],[2]) is used. One grid covers the North Atlantic from the Equator to the Bering Straits. It is rotated through 90°, so that the model's north pole is over land, and it meets the second grid, which covers the rest of the world, orthogonally at the Equator. The calculation of the grids for a given resolution is done once off-line, so the only complications to the code are the need to communicate differently where the grids meet (section 2.2) and the use there of altered finite difference coefficients.

The main timestep calculation is carried out in subroutine step. The sequential form of this routine is shown in Figure 1(a). The i and j loops are over longitude and latitude. The baroclinic call is over the column of points associated with each (i, j) point and the barotropic call is just for the surface point.

```
do baroclinic time step
    do j loop
    do i loop
        call baroclinic(i,j) (over depth)
    end i j loops

    do barotropic time step
        do j loop
        do i loop
            call barotropic(i,j) (surface)
        end i j loops

    end barotropic time step
end baroclinic time step
```

a) Sequential

```
do baroclinic time step
    do l=1, no. of points held
        find i and j
        call baroclinic(i,j) (over depth)
    end l loop
    Exchange halo information for all levels
    do barotropic time step
        do m=1, no. of points held
            find i and j
            call barotropic(i,j) (surface)
        end m loop
        Exchange halo information at surface
    end barotropic time step
end baroclinic time step
```

b) Parallel

Figure 1: Structure of the main timestepping routine (step) in the sequential and parallel codes

2.1 Grid Partitioning

Each processor is assigned a compact region of the ocean surface of either grid 1 or grid 2. As land regions have no calculations associated to them, only sea points are considered. For an efficient code, the total amount of work to be done needs to be assigned as evenly as possible between the processors. To allow this, the OCCAM code is written for irregular shaped processor regions. However, for fine resolution models, memory requirements restrict the maximum dimensions of the processor regions, at the expense of load balancing. The partitioning is done off-line and a processor map is read in at start-up.

The parallel version of subroutine step is shown in Figure 1(b). Each processor makes a list of the sea points it holds and loops over these for the baroclinic and barotropic calculations.

2.2 Communication structure

The finite difference discretisation in the horizontal direction uses a nine point stencil. Each processor has a domain consisting of a core region surrounded by a ring of 'inner halo' points, which in turn is surrounded by a layer of 'outer halo' points. The outer halo

points are points which lie in another processors calculation domain and the inner halo points are those required by other processors. The processor calculates information for the core region and the inner halo points, using the information stored in the outer halo if necessary. At the end of a time step, the updated inner halo points are sent out to the processors which need them and the new values for the outer halo are received. The finite difference stencil may straddle the boundary between the two grids. Points which are exchanged across the two grids are sent in a separate message at communication time.

The communication routines have been written to allow different message passing libraries to be used and to alter the main code as little as possible. When information needs to be exchanged, a call is made to a send or a receive subroutine. An integer, called the message tag, is passed down and this dictates the contents of the message (e.g. surface variables). The basic structure of each message is the same. Send messages initialise a send, put/pack the data in a buffer and send the buffer; receive messages pick up the buffer and unpack the data in the buffer. Each of these stages calls a subroutine which can be altered according to the communications library.

The OCCAM model is being run on a Cray T3D with the SHMEM message passing library. The shmem_put command takes data from one processing element (PE) and places it in the memory of another one but it does not check that the variable is not being used by the receiver. This may lead to cache incoherence. To avoid this, a system of buffers and locks is used around the SHMEM calls.

For the initialisation stage of sending a message, the sender invalidates its cache and searches through the buffer lock array for an empty buffer, repeating this until it is successful. The buffer is filled with the relevant data and a buffer lock for that buffer is set. Using an atomic swap, the sender gains control of the receiver's queue lock. Once successful, no other PE can gain control of the queue array or the queue length. The sender checks the length of the queue array, aborting if the queue is full as an error will have occurred. The sender's processor number, the buffer number, the message length and the message tag are placed in the queue array using shmem_put and the queue length is incremented. An atomic swap is used to release the queue lock, marking the end of the sender's responsibility.

When receiving a message, the receiver invalidates its cache and gains control of its own queue lock by using an atomic swap. The PE sweeps through the queue until it finds a message tag which agrees with the one it is looking for. Using the processor and buffer numbers held in the queue array, the receiver uses shmem_get to pick up the data from the sending PE. It then reduces the length of the queue by 1, resets the sender's buffer lock for the buffer it has picked up and releases its own queue lock, using an atomic swap. The buffer is then unpacked into the appropriate variables.

Currently the message passing is synchronous in that all halo information is sent and then received. An asynchronous implementation is being introduced where the messages are sent and then the processors continue calculating the core region for the next time step. When they reach the calculation for the inner halo points the messages are received.

2.3 I/O

Each run of an ocean model produces large quantities of data, which are used to monitor the development of the world's oceans, to analyze the ocean's behaviour and to restart the model at a later date. Similarly, at the beginning of a model run, a large amount of data is needed to start the model correctly. To simplify the handling of the data, one processor has been set aside to do all the external I/O. At the beginning of each run, this processor reads in values for restarting the model, the processor map, the topography (depths) map, and forcing data. Later in the run it also controls and handles the writing out of model results and the calculation of diagnostics. When a processor reaches an archive timestep it buffers the solution it holds before continuing with the main calculation. The I/O processor takes a slab of latitude lines and, by asking for data from the other processors, gradually fills up the slab. It then outputs the slab before moving on to the next one. In this way, the archive file is contiguous and can easily be used for analysis and for restarting the model, even if a different processor map is being used.

3 Initial Run

The model was run for three years at a resolution of $\frac{1}{4}^\circ$ and with 36 depth levels on a 256 PE Cray T3D. A 15 minute timestep was used for the baroclinic calculation and an 18 second one for the barotropic mode. The ocean temperature and salinity were relaxed to the annual Levitus data set on an annual timescale at depth and to the monthly Levitus data sets on a monthly timescale at the surface. The ocean surface was also forced using monthly ECMWF wind stresses. For these three years the Bering Straits were closed and the Pacanowski and Philander vertical mixing scheme [9] was used from day 480.

During the initial three years various problems have occurred with the numerical aspects of the code. After a few days, what seemed to be standing waves, which were increasing in size, appeared off the Pacific coast of America. These were actually surface gravity waves which were moving their own wavelength every baroclinic timestep. To overcome these, a time averaged scheme is used for the free surface calculation. For each baroclinic timestep, the free surface is stepped forward the equivalent of 2 baroclinic timesteps and the average of the free surface height and surface velocity fields are used as the solution.

Another problem was chequerboarding in areas where the barotropic flow was changing rapidly. This was overcome by adding a diffusion term to the free surface height. Laplacian operators with stencils of the form

$$\begin{bmatrix} 0 & X & 0 \\ X & X & X \\ 0 & X & 0 \end{bmatrix} - \begin{bmatrix} X & 0 & X \\ 0 & X & 0 \\ X & 0 & X \end{bmatrix}$$

were used, where X denotes a contribution from a point and 0 denotes no contribution. These stencils were altered near land masses. This addition filters out small waves and two grid-point noise. To reduce computational cost it is carried out on the first free surface timestep of each baroclinic one.

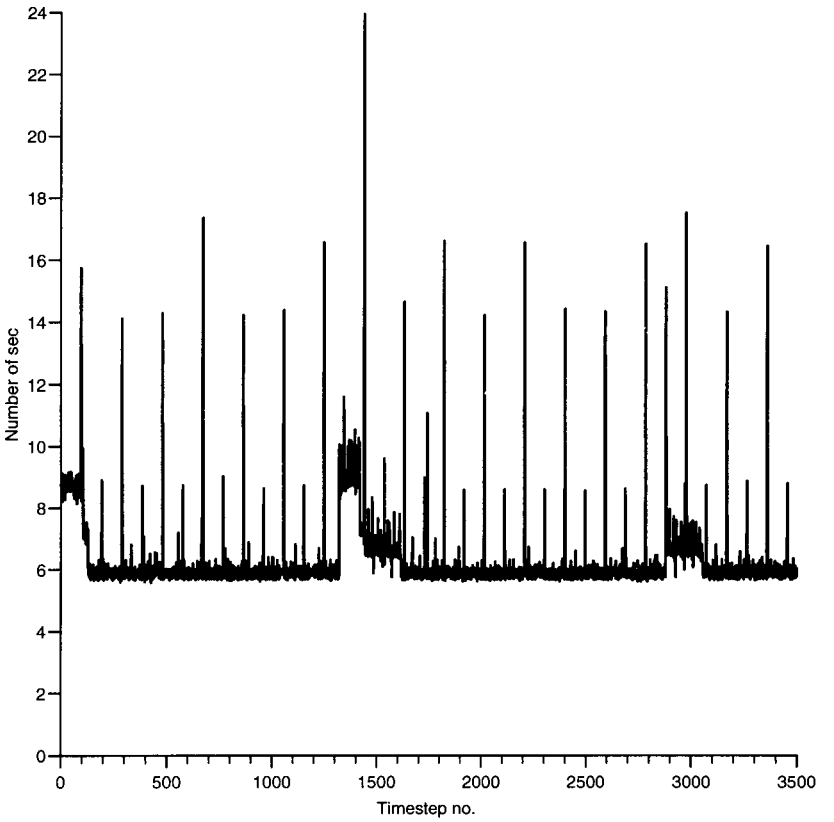


Figure 2: Number of seconds spent in each call to subroutine step for a 256 processor run.

The time taken in subroutine step for each processor of a 256 PE run is shown in Figure 2. For most timesteps, the time taken is 6 seconds. Every day (i.e. every 96 timesteps) a Matsuno timestep is used instead of a leapfrog to ensure stability. This requires two sweeps of subroutine step and accounts for the smaller peaks. Every 2 days a snapshot is taken of some of the data. Variables from two depth levels are output to a data file. Every 15 days a full restart data set is output. During these times the I/O processor is busy retrieving the data from the other processors. As all processors barrier at the top of the baroclinic timestep loop, time is spent waiting if I/O is being done (although the I/O processor does return to the top of the loop at regular intervals). The result is that while a data set is being written the time in step increases. The large peak of 24 seconds is partly because of the I/O problem and partly the time taken by the calculating processors to buffer the archive solution.

4 Future Work

The model will continue to be spun up for a further three years but with the Bering Straits open and with a fresh water flux to simulate the change in surface height due to precipitation and evaporation. After the end of the fourth model year the depth relaxation to Levitus will be removed and the surface relaxation will be weakened. ECMWF heat fluxes will be used as well as the wind fields for the surface forcing. After six years the model will be run without relaxation to Levitus. Instead, improved full surface forcing with global wind, precipitation, evaporation and heat fluxes will be used. These fluxes are being developed by a team at the James Rennell Centre, under Dr. P. Taylor.

The advection scheme used in the model is known to cause under and overshooting of the tracer (salinity and temperature) values where large gradients in tracers and/or velocities occur. This has led to sea surface temperatures of -5°C at the front between cold and warm water masses. Although these disappeared, improvements in the advection scheme need to be made for correct representation of such fronts. The QUICK scheme ([6],[8]), a second order upwind method , for the advection terms in the tracer equations will be implemented from the end of year four of the model run. The scheme requires extra halo rows (one inner and one outer) to be added. The communication for this is currently being tested.

To date very little optimisation work has been carried out. When the QUICK scheme has been implemented, investigation of various known problem areas will be undertaken. The current development of the asynchronous message passing will remove some of the contribution to the large peaks in Figure 2. Inlining of the subroutine calls in the send and receive message routines will help reduce the time spent in message passing. Most of the time in the code is spent in the free surface subroutine. This is a small piece of code and it is therefore an ideal candidate for optimisation. The T3D has a small cache and cache misses lead to a lack of efficiency. For the current case, with only 1 halo row, investigation has shown that this is not a problem. For the 2 halo case, however, cache misses will become important.

Other improvements include the use of variable bottom topography from the end of year four. This will improve the representation of sills, continental shelves and very deep ocean basins. The extension of QUICK to the momentum equations will be investigated and a sea ice model will be added. When these improvements have been made it is hoped that the resolution will be increased to $\frac{1}{6}^{\circ}$.

5 Acknowledgements

The work represented here has been carried out by the OCCAM core team, consisting of D. J. Webb, B. A. de Cuevas, A. C. Coward and C. S. Gwilliam, in conjunction with M. E. O'Neill and R. J. Carruthers of Cray Research, U.K.

References

[1] K. Bryan. *A Numerical Method for the Study of the Circulation of the World Ocean.* J. Comp. Phys., **4**:347–376, 1969.

[2] A. Coward, P. Killworth and J. Blundell. *Tests of a Two-Grid World Ocean Model.* J. Geophys. Res., **99**:22725–22735, 1994.

[3] M. D. Cox. *A Primitive Equation, 3-Dimensional Model of the Ocean.* Technical Report 1, GFDL Ocean Group, GFDL/NOAA, Princeton University, Princeton, N.J. 08542, U.S.A., 1984.

[4] E. Deleersnijder, J.-P. Van Ypersele and J.-M. Campin. *An Orthogonal, Curvilinear Coordinate System for a World Ocean Model.* Ocean Modelling, **100**, 1993. Unpublished ms.

[5] M. Eby and G. Holloway. *Grid Transform for Incorporating the Arctic in a Global Ocean Model.* Climate Dynamics, **10**:241–247, 1994.

[6] D. Farrow and D. Stevens. *A New Tracer Advection Scheme for Bryan and Cox Type Ocean General Circulation Models.* J. P. O., **25**:1731–1741, July 1995.

[7] C. S. Gwilliam. *The OCCAM Global Ocean Model.* Coming of Age (The Proceedings of the Sixth ECMWF Workshop on the use of Parallel Processors in Meteorology). Editors Geerd-R Hoffman and Norbert Kreitz, World Scientific, 1995.

[8] B. P. Leonard. *A Stable and Accurate Convective Modelling Procedure Based on Quadratic Upstream Interpolation.* Computer Methods in Applied Mechanics and Engineering, **19**:59–98, 1979.

[9] R. C. Pacanowski and S. G. H. Philander. *Parameterization of Vertical Mixing in Numerical Models of Tropical Oceans.* J. P. O., **11**:1443–1451, 1981.

[10] R. C. Pacanowski, K. Dixon and A. Rosati. *The GFDL Modular Ocean Model Users Guide.* Technical Report No. 2, GFDL Ocean Group, 1990.

[11] A. J. Semtner. *An Oceanic General Circulation Model with Bottom Topography.* Technical Report No. 9, Dept. of Meteorology, UCLA, 1974.

[12] D. J. Webb. *An Ocean Model Code for Array Processor Computers.* Technical Report No. 324, Institute of Oceanographic Sciences, Deacon Laboratory, 1993.

[13] D. J. Webb. *A Multi-Processor Ocean General Circulation Model using Message Passing.* Published by Southampton Oceanography Centre, 1995.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Concurrent Distributed Visualization and Solution Steering

Robert Haimes
Principal Research Engineer, Department of Aeronautics and Astronautics
Massachusetts Institute of Technology, Cambridge, MA 02139
haimes@orville.mit.edu

1. Introduction

To view the results of a Computational Fluid Dynamics (CFD) solver, as the solution progresses, has many benefits. If this window to the solution is through a visualization system designed for CFD class problems, then that allows debugging of steady-state solvers at a macro scale. Solution quality assessment and run-time intervention are possible if the current results are easily accessible. Also, co-processing visualization of the results of transient applications relieves many resource limitations posed by these cases. Solution steering is, of course, possible if one can watch the results of an unsteady simulation as they progress.

Though it has always been a dream of many CFD researchers, almost no 3D transient codes allow solution steering. This is because the following difficult issues:

Computational arena. 3D transient CFD codes require super-computer class hardware (which includes parallel machines) in order to get results in a timely manner. Any steering scheme that cannot be run with these machines will not be used. Suggested steering techniques (such as Burnett et. al. [1]) that require recasting the solver in new language assume that the resources required to generate the data are small.

Tool-kit. Mating the visualization to the solver must be simple but general. The visualization tool-kit should be a complete CFD viewer. Vaziri and Kremenetsky [2] discuss using CM/AVS on a TMC CM-5 for watching the results from a 3D simulation. Steering is possible by the use of constructed widgets within the AVS interface. This work, though innovative, cannot be generally applied. The technique requires a CM-5 and the entire scheme suffers from the data handling inefficiencies associated with AVS.

Coupling. The steering scheme must be able to fit in, or adapt, to the solver. Usually a great deal of effort goes towards making the solver run efficiently. The visualization of the results must be subservient to the solution scheme and easily coupled.

When discussing transient applications within this paper, the following classification is used:

Data Unsteady: In this type of application the grid structure and position are fixed in time. The data defined at the nodes (both scalar and vector) changes with each time step. An example is when a boundary condition in the domain is changing.

Grid Unsteady: These cases are 'Data Unsteady' plus the grid coordinates associated

with each node are also allowed to move with each snapshot. An example of this is stator/rotor interaction in turbomachinery. The stator and rotor grids are separate, with the rotor grid sliding past the stator grid. In this case the stator portion is actually 'Data Unsteady' and the rotor grid moves radially.

Structure Unsteady: If the number of nodes, number of cells or cell connectivity changes from iteration to iteration the case is 'Structure Unsteady'. An example of this mode is store separation.

2. Traditional Computational Analysis

Before discussing a complex system built on concurrent visualization, it is important to view how traditional analysis using CFD is performed. The computational steps traditionally taken for analysis (or when CFD is used in design) are:

- Grid Generation. The surfaces of the object(s) to be analyzed (usually derived from a CAD system) specify part of the domain of interest. Usually for the analysis of external aerodynamics, the airplane is surrounded by other surfaces that extend many body lengths away from the craft. This enclosed volume is then discretized (subdivided) in one of three ways. Unstructured meshes are built by having the subdivisions composed of tetrahedral elements. Another technique breaks up the domain into sub-domains that are hexahedral. These sub-domains are further refined so that individual cells in the block can be indexed via an integer triad. The structured block schemes may have the sub-domains abut or overlap. Finally, some systems use meshes that are not body fit. In this class of grid generation, the surfaces of the object intersect a hierarchical (embedded) Cartesian mesh. For all of these methods, the number of nodes produced for complex geometries can be on the order of millions.

- Flow Solver. The flow solver takes as input the grid generated by the first step. Because of the different styles of gridding, the solver is usually written with ability to use only one of the volume discretization methods. In fact there are no standard file types, so most solvers are written in close cooperation with the grid generator. The flow solver usually solves either the Euler or Navier-Stokes in an iterative manner, storing the results either at the nodes in the mesh or at the element centers. The output of the solver is usually a file that contains the solution. Again there are no file standards. PLOT3D [3] files can be used by grid generators and solvers that use structured blocks and store the solution at the nodes. In practice, this file type is not used by the solver writer because it contains no information on what to do at the block interfaces (the boundary conditions to be applied). Therefore, even this subclass of solvers writes the solution to disk in a manner that only the solver can use (to restart) and this file is converted to a PLOT3D file for the next step.

- Post-processing Visualization. After the solution procedure is complete, the output from the grid generator and flow solver is displayed and examined graphically by this step in this process. Usually workstation class equipment with 3D graphics adapters are used to quickly render the output from data extraction techniques. The tools

(such as iso-surfacing, geometric cuts and streamlines) allow the examination of the volumetric data produced by the solver. Even for steady-state solutions, much time is usually required to scan, poke and probe the data in order to understand the structure of the flow field. In general, most visualization packages read and display PLOT3D data files as well as a limited suite of output from specific packages. There are some systems that can deal naturally with unstructured meshes and few that can handle hybrid structured/unstructured solutions. No general package can, at this point, deal with the output from Cartesian systems.

This 3-step process has proceeded for years as individuals and teams have worked on each component with the assumption that a steady-state problem is being tackled. Therefore this process works (sometimes efficiently) for steady-state solutions. When these steps are applied to a transient problem there are two crucial steps that easily become overwhelmed. First, if the grid changes in time, the grid generation process can take enormous amounts of human interaction.

Understanding the results is the second limitter to effectively using the traditional CFD analysis steps for transient problems. What is usually done is the production of a movie. This is accomplished by treating each saved snap-shot of the solution (on disk) as a steady-state visualization task. Image (window) dumps are produced from the same view (or a fly-by) with the same tools active. These images are played back a some later time to give the viewer information on what is changing within the solution. This can be a very efficient method for communicating what is going on in the changing flow field, but is non-effective as a vehicle for understanding the flow physics. Rarely are we smart enough to 'story board' our movie in a way that displays salient features without finding them first.

3. Problems with Post-processing

If we follow the traditional computational analysis steps for CFD (assume the simple case - 'Data Unsteady'), and we wish to construct an interactive visualization system, we need to be aware of the following:

- Disk space requirements. A single snap-shot must contain at least the values (primitive variables) stored at the appropriate locations within the mesh. For most simple 3D Euler solvers that means 5 floating-point words. Navier-Stokes solutions with turbulence models may contain 7 state-variables. The number can increase with the modeling of multi-phase flows, chemistry and/or electo-magnetic systems. If we examine a 5-equation system with 1 million nodes (the field variables stored at the nodes) a single snap-shot will require 20 Megabytes. If 1000 time-steps are needed for the simulation (and the grid is not moving), 20 Gigabytes are required to record the entire simulation. This means that the workstation performing the visualization of this simulation requires vast amounts of disk space. You do not want to access this much data over a distributed file system!

- Disk speed vs. computational speeds. The time required to read the complete solution of a saved time frame from disk is now longer than the compute time for a

set number of iterations from an explicit solver. Depending on the hardware and solver an iteration of an implicit code may also take less time than reading the solution from disk. If one examines the performance improvements in the last decade or two, it is easy to see that depending on disk performance (vs. CPU improvement) may be a bad 'bet' for enhancing interactivity. Workstation performance continues to double every 18 months. The price per Megabyte of disk drives has dropped at amazing rates, but the performance of commodity drives has gone from about 1 Megabyte/sec in 1985 to about 5 Megabytes/sec in 1995. To augment disk performance, techniques may be applied like disk striping, RAID technology, or systems like Thinking Machine's DataVault, which depend on using multiple disks and controllers functioning in parallel. But most workstations currently have SCSI interfaces that limit data transfer rates to about 5 Megabytes/sec (SCSI II) per chain. High performance workstations that have other methods may only be able to move 20 Megabytes/sec through an I/O channel. Therefore, if you wish to post-process on a normal workstation, it may take 4 seconds per iteration, just to read the solution for the above example.

- Cluster and Parallel Machine I/O problems. Disk access time is much worse within current Massively Parallel Processors (MPPs) and cluster of workstations that are acting in concert to solve a single problem. In this case we are not trying to read the volume of data, but are running the solver to produce the data. I/O is the bottleneck for an MPP with a front-end. The MPP probably has the ability to compute in the GigaFLOP range but all the data has to be funneled to a single machine and put on disk by that machine. Clusters of workstations usually depend upon distributed file systems. In this case the disk access time is usually not the bottleneck, but the network becomes the pacing hardware. An IBM SP2 is a prime example of the difficulties of writing the solution out every iteration. The machine has a high-speed interconnect, but the distributed file system does not use it. There are other access points into each node. Most SP2s have an Ethernet port for every node, some also have FDDI connections. These traditional network interfaces must be used for the file system. The SP2 can also be used in the traditional front-end paradigm if one of the nodes has a disk farm. In this model, the high-speed interconnect can be used with explicit message passing to the I/O node that does the writing. This obviously requires special code and knowledge of the underlying hardware. In our above example, it would take about 20 seconds to write one time frame (Ethernet hardware - distributed file system). If the machine is dedicated (no other tasks running), then that wastes 20 seconds of cycles.

- Numerics of particle traces. Most visualization tools can work on a single snap shot of the data but some visualization tools for transient problems require dealing with time. One such tool is the integration of particle paths through a changing vector field. After a careful analysis (numerical stability and accuracy) of integration schemes [4], it has been shown that there exist certain time-step limitations to insure that the path calculated is correct. Even for higher order integration methods, the limitation is on the order of the time step used for the CFD calculation. This is because of a physical limit, the time-scale of the flow. What this means for the

visualization system is that in order to get accurate particle traces, the velocity field must be examined close to every time step the solver takes.

Because of the disk space requirements and the time to write the solution to disk, the authors of unsteady flow solvers perform some sort of sub-sampling. This sub-sampling can either be spatial or temporal. Because the traditional approach is to deal with the data as if it were many steady-state solutions, this sub-sampling I/O is almost always temporal. The individual running the simulation determines the frequency to write the complete solution based on the available disk space. In many cases, important transitions are missed. Also since the solution is coarsely sampled in time, streaklines (unsteady particle paths as discussed above) almost always produces erroneous results. The problem with sub-sampling is that the time-step selected for the visualization is based on the available disk space and not the physical problem.

With the huge storage equipment (and financial) burden on the compute facility it is no wonder that only large national labs routinely visualize results from transient problems. We must adopt another visualization architecture in order to overcome the limitations produced by post-processing.

4. Co-processing Visualization

One solution to the problems described above is co-processing. Concurrent solving with interactive visualization can relieve the compute arena of the disk space, speed and sampling issues. It does require a complete re-thinking of the architecture of the visualization system and poses the following issues:

Coupling to the solver. The solver must communicate with the visualization system. This can be accomplished by one of three methods:

Disk files: In this approach the solver task communicates with the visualization task by data written to disk. This method is rejected for the disk timing and I/O arguments discussed above.

Shared memory: In this approach the solver and visualization system communicate via shared memory. This has the advantage that the solver and visualization tasks are separate and the communication is fast. The disadvantages are that the solver must be written with the shared memory interface, and the data that is exposed must be done in a way that the visualization task knows where and how to get individual data elements. Also, some method is required to mark the data invalid as it is being updated.

Application Programming Interface (API): This method couples the visualization task (or some portion of the visualization system) with the solver. This coupling is done at the programming level. The advantages to this approach are that all the data is shared (there is only one task), no special system level requirements are needed, and it can be used with solvers written in different languages. The challenge is to develop a software architecture that is general, non-invasive and that allows the visualization system and solver to function independently.

Additional resources. Whichever approach is selected, an additional burden is placed on the computer resources. Now both the solver and at least a portion of the visualization system are active on the computer. The visualization system should not place a large encumbrance on either the memory requirements or need large numbers of CPU cycles.

Robustness. The visualization system is running concurrently with the solver. Either it is part of the same task, or has access to crucial data. The visualization must be robust and not interfere with the solver's functioning. A problem with the visualization portion must not crash the solver.

Interactivity. It is important for interactivity that the resources required by a co-processing visualization system be a minumum. The cost of CPU cycles and additional memory are the barometer for the frame rate. To accomplish fast frame rates, it is necessary to use the classification of transient problems already discussed. If the visualization system is properly designed, this classification can be used to determine when the results of a tool should be saved and when recalculation is necessary.

5. Co-processing with pV3

The distributed visualization tool-kit, **pV3** [5], has been developed for co-processing. **pV3** builds heavily from the technology developed for **Visual3** [6]. This re-tooling was necessary to support the visualization of transient problems without having to dump the entire volume of data to disk every iteration. The design of **pV3** allows the solver to run on equipment different than the graphics workstation.

pV3 segregates the visualization system into two parts. The first part is the task that actually displays the visualization results onto the graphics workstation, the server. The second portion is a library of routines that allows the solver (solver portions or solvers) to communicate with the visualization server by providing windows to the data at critical times during the simulation. This client library separates all visualization based calculations from the flow solver, so that the solver programmer only needs to feed **pV3** the current simulation data.

pV3 has been designed to minimize network traffic. The client-side library extracts lower dimensional data required by the requested visualization tool from the volume of data in-place. This distilled data is transferred to the graphics workstation. To further reduce the communication burden posed by the visualization session, the transient problem classification described above is used. Only the extracted data that has changed from the last iteration is sent over the network.

An added benefit of this architecture is that most of the visualization compute load is run in parallel for a cluster environment or an MPP. This is because most visualization tools are based on operations performed on individual cells within a partition. This means that these tools are embarrassingly parallel. If the partitions are balanced on the basis of cells, then the result of the visualization computation is also executed in parallel with some balance. Therefore much better performance for the visualization is achieved.

Each client may be assigned a different class of unsteadiness. This tells the system if data needs to be recalculated and re-transmitted. For example, if a sub-domain is 'Data Unsteady' and a geometric cut is desired (such as a planar cut), only the field data at the nodes (scalar and/or vector) need to be re-transmitted to the server every iteration. The

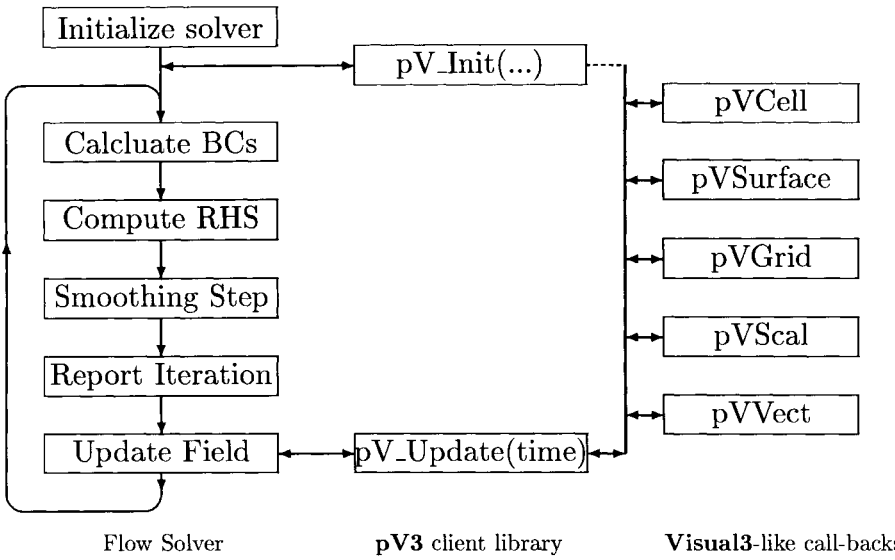


Figure 1. Archetypical Iterative CFD Solver coupled with **pV3** – A Client

geometry (node locations and mesh data to support the surface) is communicated only when the cut is initiated. If, in the above example, the case is 'Grid Unsteady' all data needs to sent every iteration to the server.

To maximize **pV3**'s effectiveness, client-side libraries exist for most major workstation vendors, MPPs and super-computer manufacturers. **pV3** has been designed so that the process of coupling to the solver is simple and noninvasive. The solver's control structure is maintained with the visualization subservient. In the distributed setting, client(s) perform their appropriate calculations for the problem, then as the solution is updated, supplied call-back routines are executed by the visualization library.

The coupling is performed by adding two calls to the **pV3** client library in the solver code. The first call informs the visualization system about the following:
Volume discretization – the number of disjoint elements, their type, and the number and size of structured blocks.
Unsteady mode – 'Steady-state', 'Grid', 'Data' or 'Structure Unsteady'.
Visualization field variables – the number, type, name and limits for each scalar, vector and threshold field.
Programmed cuts – Any non-planar geometric cuts defined for this case.

After a successful initialization, the process is entered into the **pV3** client group.

The programming model for a typical iterative CFD application can be seen in Fig. 1 on the left. In general, at the top of each time step boundary conditions are calculated and set, the solver computes the change to the solution for the entire volume. Then the change to the solution is applied. The two access points into the solver are depicted in the middle. The call-backs required by **pV3** are shown toward the right of the Figure.

This second access point (call to PV_UPDATE) informs the visualization system of two things; (1) the simulation time for time-accurate solvers and (2) the solution has been updated and synchronized. This is the mechanism that allows **pV3** to operate. Without the server running, every time the solution is updated and this call is executed, a check is made for any members in the server group. If none is found, this call returns. When the visualization task starts, it enrolls in the server group, and waits for initialization messages from the clients. The next time the client goes through an iteration, and calls the **pV3** update routine, an initialization message is sent to the server. The server processes all initialization messages, figures out the number of clients and the visualization session begins. Each subsequent time the client goes through the loop and makes this **pV3** call, visualization state messages and tool requests are gathered. Then the appropriate data is calculated, collected and sent to the graphics workstation.

Because PV_UPDATE is where all requests with the graphics workstation are performed, the overall response to state changes and the interactive latency associated with these changes (and the next time frame) depends on how often this routine is called. About one call per second is optimal. If the solution is updated significantly faster then much of the compute cycles will be used for the visualization, moving the solution slowly forward in time. In this case it is advisable to call PV_UPDATE only every N times the solver updates the solution. (When skipping iterations and particle integrations are desired, some analysis of how many iterations can be missed is required.)

The more difficult case is when the solution update rate is much slower than optimal. In this situation, there are two choices; (1) live with large lags between user requests and the result of the request seen on the screen or (2) setup another task between the solver and the **pV3** server. This software's responsibility is to communicate with the solver. It should be the task to make all the **pV3** client-side calls. This secondary task can communicate with the solver using PVM (and therefore must be on the same machine to avoid large network transfers). Or, if the machine supports multi-threading, the task can be a second thread and perform double-buffering of the solution space, so no data need be transferred. These methods are a trade-off of memory usage for interactivity.

When the user is finished with the visualization, the server sends a termination message, leaves the server group, and exits. The clients receive the message and clean up any memory allocations used for the visualization. The **pV3** client update call reverts to looking for members in the server group.

The visualization system gets the solver's data via call-back routines supplied with the solver. These routines furnish details about the volume discretization, define the surfaces (partition as well as computational boundaries), the node locations and the requested scalar and vector fields. These are virtually the same call-backs used for a **Visual3** driver.

During the design of **pV3**, the state of the solver (client side) was always viewed as the most important component. No error condition, exception or unusual state within the visualization portion of the client effects the solver. To maintain this robustness, no visualization state is stored within the client, all tools are handled via a specific request for data and the response is calculated and sent to the server. If any condition occurs that would cause some problem with the visualization, the **pV3** server shuts itself down.

This allows the simulation to continue, with at most a delay of **pV3**'s time-out constant. The **pV3** server can always be restarted to re-connect to the solver at any later time.

6. Steering with pV3

Before using **pV3** for solution steering, it is important to understand the dynamics of this distributed system. The server is data double-buffered. This is done for interactivity and allowing the view only of complete data frames. As data from the current time-step is collected, the investigator is viewing the last iteration. When all the data from the current iteration has been received by the server, the buffers are flipped and the extracts from the new iteration are viewed. It should be noted that by the time the server has completed a time frame, the solver is already on the next iteration. This means that what is being viewed is, at best, an iteration behind the current solution.

The mechanism used within **pV3** for communication (and therefore steering) between the server and the solver is simple. It uses the architectural framework already in place on the client side. If the individual sitting in front of the graphics workstation wishes to send a command to the solver, then a character string is typed into the server. This string is sent to all active clients. When, within the request handling of PV_UPDATE, this string is encountered, a call-back routine is executed. The **pV3** client-side library contains a dummy string-catch routine, so that if the call-back is executed without a mechanism to interpret these messages, nothing is done. If the solver's programmer wishes to allow communication with the **pV3** server, all that is required is that the call-back be supplied. There are also routines so that character strings can be output on the server's console to give feedback or indicate errors in parsing the command strings.

All communication is between the server and the known **pV3** clients. It is important that if all solver tasks are not clients, that these non-client tasks are also passed the steering information. This is an issue if the solver is set-up using the master/slave paradigm common within PVM applications. In most cases, the master will not be a **pV3** client, for it usually does not contain a volume partition. It is the responsibility of a slave-client to pass any changes back to the master. This may require additional message passing not needed within the normal solver and **pV3** usage.

Using this technique, the complex control of a running application is made simple.

7. Conclusions

The computational analysis of such complex problems as the flow about complete aircraft configurations has yet to reach its full potential even for steady-state problems. Euler and Navier-Stokes codes are yet to be used routinely in the design cycle. This is partly due to the difficulties in moving data (via non-existent file standards) between the traditional components of the analysis steps. These data handling problems become intractable for transient applications.

One obvious solution to dealing with the masses of information created by transient CFD codes is to closely couple the components avoiding placing data on disk unless necessary. This scheme also has some difficulties, but the direction that computer hardware is taking leads one to believe that this is the best approach. An added benefit of any co-processing is the ability to remove steps from the analysis (and therefore the design)

loop, potentially making the entire process streamlined. Also if one is watching the results of the solution procedure as they are calculated, solution steering is possible.

pV3 is an attempt to closely couple 3D CFD solvers with a complete visualization system for transient applications. This marries only the last two phases of the traditional computational analysis. As grid generators become more reliable and require less intervention, it is possible to envision completely coupled systems that incorporate, with a seamless Graphical User Interface, all three components. Several commercial vendors now supply turn-key integrated products for steady-state applications.

It is the view of this author, that in the near future, it will be possible to 'fly' a new design completely within the computer. Geometry changes will be made and propagated throughout the system. This is steering beyond modifying the boundary conditions of the solver. With this technology, a designer will be able to test a new configuration not just in steady-state, but in off-design situations. Through interactive investigation, the issues of design (are the control surfaces in the right place and are they sized correctly), issues of maneuverability (does the craft respond properly to control), and side effects (vortices shed, noise) can be addressed.

pV3 has shown that it is possible to perform concurrent solver/visualization for transient problems. The dream of complete flight simulation is possible at the solver level because the compute power continues to increase greatly. Grid generators, with the input of 'good' geometry can generate meshes about complex 3D geometries on the order of 10 minutes on currently available workstations. If small and isolated geometry changes are required, the grid modification time can be much less. The key component, that is now missing, is the control over the geometry, including changes and movement to control surfaces. With this control, and a smooth interface to the gridding component, a new future in numerical simulation for CFD is possible.

REFERENCES

1. M. Burnett, R. Hossli, T. Pulliam, B. VanVoorst, and X. Yang. Toward Visual Programming Languages for Steering Scientific Computations. IEEE Computational Science and Engineering, 1994.
2. A. Vaziri and M. Kremenetsky. Visualization and Tracking of Parallel CFD Simulations. Proceedings of HPC '95, Society of Computer Simulation, 1995.
3. P. Buning and J. Steger. Graphics and Flow Visualization in Computational Fluid Dynamics. AIAA Paper 85-1507, 1985.
4. D. Darmofal and R. Haimes. An Analysis of 3-D Particle Path Integration Algorithms. AIAA Paper 95-1713, 1995.
5. R. Haimes. **pV3**: A Distributed System for Large-Scale Unsteady Visualization. AIAA Paper 94-0321, 1994.
6. R. Haimes and M. Giles. **Visual3**: Interactive Unsteady Unstructured 3D Visualization. AIAA Paper 91-0794, 1991.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

From R&D in parallel CFD to a tool for computer aided engineering

W. Loeve

National Aerospace Laboratory, P.O. Box 90502, 1006 BM Amsterdam, The Netherlands

For industry, CFD is only of interest if it helps to minimise the cost and maximise certainty and speed for a "good" design. Consequently a performance of about 10 GFLOP/s is required for R&D and application of the next generation CFD tools that are of interest for aircraft industry. This requires parallelization of CFD. Technically and financially for the time being vector computers with the maximal possible performance per processor are the best compute servers for CFD. The compute servers have to be integrated in the IT infrastructure of the user organisation. The integration shall be such that for use and management of data, software and electronic documents, the IT infrastructure presents itself to the users as one single virtual computer.

1 INTRODUCTION

Many R&D organisations are confronted with reduced budgets and more critical evaluation of R&D targets and results by management or funding agencies than in the past. These changes force R&D organisations to reconsider their base of existence. A sound strategy for continuation of R&D is to derive R&D targets more directly from economic targets. The economic targets are dictated by industry.

In view of these considerations, R&D in CFD should be directed towards realisation of tools that can be used in industrial engineering processes. In that case it is essential at least to minimise the time-to-market and cost of new CFD tools. Quality of new CFD tools in this context is determined by the degree of satisfaction of the industry for which the tools are meant. Innovation and technology are of minor importance in the quality measure for new CFD tools than the added value for the using industry.

From listening to industry, the following considerations emerge:

1. Industry is confronted by drastic changes in their commercial marketplace. Customer satisfaction has become the main issue in industry. R&D of CFD therefore is only of interest for industry if it leads to new CFD tools that can be used to increase the certainty of design processes, to speed up design processes, to reduce the cost of design processes or to generate any other benefit that helps to satisfy the aircraft operator.
2. For industry not only CFD computations are of interest. New computational methods are judged with respect to a possible improvement of the complete CFD process: geometry handling, grid generation, configuring the computation for the applied computer, computation, analysis of results and last but not least the exchange of information with other processes in designing aircraft. Moreover it must be possible to implement the tools in the industrial information technology (IT) infrastructure.

3. Complexity of the computer networks and of the collections of software systems used in engineering processes both increase continuously in industry. The IT infrastructure requires tools for management of data, software and electronic documents. Because aircraft design is cooperative work, sharing of information between specialists is required. Tools to support this also have to form part of the IT infrastructure. The CFD software needs to operate together with the tools.
4. Industry requirements with respect to the behaviour of the software elements used in the CFD process not only concern nominal application conditions. Requirements with respect to behaviour under non-nominal conditions (user or system generated errors) and time and cost for maintenance are just as important. CFD software has to be designed in view of error handling, testability, maintainability and re-usability.

NLR operates as support organisation for aerospace industries. As such, NLR provides facilities for experiments and for computing. Software is being developed for CFD and NLR's IT infrastructure contains compute servers for this. These servers are accessible for interested parties such as industries. NLR's position in the market requires the availability of among other things maximum computer performance for R&D and application of CFD. The considerations described above are regarded as quality criteria for CFD. They have been the guidelines at NLR for the choice of:

- the type and performance of the compute servers for CFD,
- tools for integration of the compute servers in the IT infrastructure of NLR, and when desired in that of customer organisations,
- practices for management and engineering of software systems.

In the next chapters this will be discussed.

2. CHOICE OF COMPUTE SERVER FOR CFD

When we look at the most powerful compute servers installed in the world [1], we see that in recent years, at the top of the list, parallel computers based on many scalar processors and distributed memory are taking over in number from the vector computers with several processors and shared memory. NLR and its customers are solution oriented organisations as far as application of computers is concerned. Cost and time-to-market of new software have to be minimised, if necessary at the cost of innovation and technology. In view of this, for many of the users of NLR systems the massively parallel machines in the TOP500 list [1] are inadequate. Although NLR has a well developed informatics division, the capabilities of this division are used for automation in customer organisations and in NLR rather than for development of parallelization strategies on computers that require too much effort. For the time being, the massively parallel computers are regarded as interesting platforms for R&D in computer science mainly and not for application driven R&D in aircraft CFD. As a result of this conclusion, NLR has only compared vector computers and workstations with regard to suitability for CFD, as part of the procurement process of a new computer.

The CFD application in aircraft design is characterised with the following:

1. The most advanced industrially applicable CFD for aerodynamic analysis at the moment is based on the Reynolds-Averaged Navier-Stokes equations [2]. One not too detailed computation for a full aircraft configuration requires about 5 hours in case of a

sustained computer performance of about 500MFLOP/s. One of the interesting applications of these computations for the near future is support of aerodynamic optimisation. See for instance publications from Boeing and NASA specialists [3]. This target requires a computer performance that is one to two orders larger than what is used in analysis. This computer performance is required for instance because about 40 times solution of the flow equations in combination with an optimisation algorithm is required.

2. The process of designing an aircraft is characterised by a large number of activities executed by specialists from various disciplines. The specialists use a variety of tools including many different software systems. Rubbert [2] describes a possible aerodynamic wing design process. In this process the Reynolds Averaged Navier Stokes (RANS) equations so far only are solved to check selected points in the aircraft flight envelope. The limited use of the RANS equations has to do with the cost involved according to [2]. In the same paper it is explained that industry needs to minimise the time for design and to maximise certainty of the design process. Apparently it can help industry if the performance and the performance/price ratio of compute servers are increased so that the RANS equations can be solved more frequently in the design process.
3. Application of CFD in industry has all the characteristics of computer supported co-operative work [2]. Also R&D of industrially relevant CFD requires a computer supported cooperative work approach [4]. This means that information management and sharing of information between specialists has to be safeguarded in the IT infrastructure of industry and of R&D organisations. The information becomes available in three forms: data, software and electronic documents.

From the description above of CFD in design in industry, the requirement can be derived that it shall be possible for more certainty and less design time, to perform one aerodynamic optimisation run in one night and to apply the Reynolds Averaged Navier-Stokes calculation more frequently than is possible at the moment. The first requires a sustained computer performance of 10 à 20GFLOP/s. If this compute power can be made available at sufficiently low price, it can also serve to realise the second requirement. This compute power has to be integrated in the IT infrastructure of industry together with other computer based servers such as workstations and data management servers to support use and sharing of information in the organisation.

Industry prefers workstation clusters for CFD because these are available in large quantities during the night and weekend. These are purchased for mainly engineering activities and are available at no cost so far for CFD according to Lewis (Boeing), Cosner (McDonnell Aircraft) and Fischberg (Pratt&Whitney) [5]. These workstations are clustered in units of about 16 for execution of CFD calculations. Limiting the number is because adding more workstations to the cluster does not result in increase of performance. Why this is the case is explained with the help of a simple application oriented model in [6].

For NLR it is essential to be informed in-depth about the situation in the computer market. It also appears that NLR is an interesting source of in-depth information about advanced computer applications for computer manufacturers. As a consequence NLR is so lucky to be able to cooperate with NEC with the purpose to enhance NEC supercomputers and to optimise application at NLR thereof. NLR also appreciates the cooperation with

IBM and SGI that generates information for NLR's system engineers about how to optimise use of multi-processor workstations and clusters for CFD. For IBM and SGI, information is generated about how to improve their systems for CFD applications. The cooperation with SGI, apart from parallel computing and networking also covers IT environments for computer supported cooperative work. The experiences from the cooperations and from in-house availability of top-of-the-line vector computers since 1987, were used to compare the suitability for industrially relevant CFD, of contemporary CRAY an NEC vector computers and SGI's (Power) Challenge workstation based high performance computer systems that also can be found in the TOP500 list [1].

Solvers for both multi-block structured grids and unstructured grids were applied to determine the suitability of the mentioned compute servers for CFD. Parallelization was based on domain decomposition. The result of the investigations is that for the applications the single processor sustained performance of the Power Challenge processors was about 25% of the peak performance of 300MFLOP/s. Of the NEC SX-3 that at present is installed at NLR, the single processor sustained performance varies between 20 and 35% of the peak performance which is 2.75GFLOP/s. The SX-3 is 7 to 14 times faster than the Power Challenge per processor. With the present performance of workstation clusters this means that the maximum sustained performance of Power Challenge single processor workstation clusters, is about the same as the sustained performance of 1 SX-3 processor, for practical aircraft CFD. This is also the case for the Power Challenge configurations with 18 processors that share memory. On the multi-processor workstations CFD applications can be implemented easier than on workstation clusters with distributed memory.

The performance of multi-processor workstations can only be improved for aircraft CFD if the transfer speed of data and the single processor sustained performance are improved considerably [7]. This will lead to computer characteristics that are comparable with characteristics of the vector computers of the NEC SX-4 and CRAY T90 type. The single processor peak performance of these computers is respectively 2 and 1.8GFLOP/s. The single processor sustained performance for aircraft CFD is about 1GFLOP/s. The vector computers that are on the market at the moment are supplied in configurations with up to 32 processors with shared memory. With these configurations it is easy to obtain the performance of 10 à 20GFLOP/s that is required for the next step in development and application of CFD in aircraft design.

Now of course it is interesting to look at prices and price/performance of the computers discussed so far. The problem with prices of computers is that they vary in time and that list prices mostly are not paid. In what follows a picture is given of list prices only at a given moment in time (and place). In april 1995 NLR was informed by NEC and CRAY that the list price of a 16 processor vector supercomputer of the type SX-4 and T90, was about NLG 20M and NLG 25M respectively. The single processor sustained performance of these systems is about 900MFLOP/s for the NLR applications.
SGI offered an 8 processor Challenge for NLG 0.8M, and an 8 processor Power Challenge for NLG 1.5M. The single processor sustained performance of these workstations appeared to be 20MFLOP/s and 70MFLOP/s respectively in the NLR tests.

The numbers lead to the conclusion that the price/performance of the vector processors in contemporary supercomputers is about twice as good for CFD computations as the price/performance of a workstation processor that dominates the bottom of the TOP500 list

[1]. The internal communication performance of vector supercomputers is much better than that of multi-processor workstations. As a result of this, the 16 processor vector supercomputers will have a sustained performance of more than 10GFLOP/s for aircraft CFD computations. The conclusion is that the multi-processor vector computers can supply the required sustained performance for application oriented CFD and that multi-processor workstations cannot. It should be remarked that for other tasks in the CFD process, workstations are the best or only choice. For grid generation for instance, the single processor NEC SX-3 performance is the same as the single processor SGI Power Challenge performance.

Based on the evaluation described above the conclusion is that R&D in parallelization of aircraft CFD for the time being best can be based on vector computers with the maximal possible performance per processor. Configuring the CFD computation for this type of computer is straightforward. To effectuate this conclusion in practice requires to avoid that all money is spent on technically and economically sub-optimal combinations of workstation processors. This is possible in research organisations. However it requires involvement of high management levels in the automation policy in organisations. NLR has a centralised automation strategy and was able to buy a 32 processor NEC SX-4 vector computer for R&D in CFD. The first 16 processors will be installed in June 1996. The second set of 16 processors will be installed 1.5 year later. Because some industries prefer workstation-based automation, at NLR the CFD codes also are ported to combinations of workstations. This is facilitated by application of an object oriented software structure.

Introduction of new vector supercomputers probably is difficult for aircraft manufacturers. This is because industry needs already so many workstations for their engineering and production activities. It is the strong belief of the author that cooperation of national aerospace institutes with aircraft manufacturers can lead to a situation that at short notice computers with the required 10GFLOP/s sustained performance can be applied in aircraft design. This will make it possible for industry to improve the certainty and speed for a "good" design. As soon as proper cost accounting for workstations is introduced in industry the use of vector supercomputers probably will also prove to lower the cost for a good design. In the next chapter some remarks will be made about the use of remote compute servers in or outside the user organisation. Despite the conclusions presented above, for the time being CFD computational methods developed in R&D organisations have to be ported to workstation clusters. This will restrict applicability of the CFD tools in the design process.

3. INTEGRATION OF CFD APPLICATION ENVIRONMENTS

For application of CFD not only high performance compute servers are required. There is also a need for graphical presentation and processing with the help of workstations and for information management by means of specific servers. In applications of CFD in integrated aircraft design, activities are performed by cooperating teams. For this reason the different computer based systems have to be integrated in a network. This network has to give access, for all persons and groups that take part in the cooperation, to the computers as well as the software, data and electronic documents that are stored on the computers. The capacity of the network has to be sufficient for the often large information

flows.

In CFD environments the easy and simultaneous processing of often distributed information is required as well as a reliable seamless flow of information through the successive steps of CFD processes. The development of software for CFD concerns physical/mathematical modelling, software engineering, and validation with relation to the relevance of results of simulation. Especially development of new advanced CFD tools requires contributions from various disciplines. As a result of this, development and application of CFD tools both require a cooperative approach, and for both a similar computer and software is required.

With increasing importance of CFD for economically relevant applications, also quality requirements for CFD environments become more severe. Quality concerns reliability, maintainability, demonstrated applicability, portability, growth potential, flexibility and time-to-market of CFD tools. This leads to the necessity to apply quality management to the engineering and production of software for CFD. As a result of this, characteristics that are required for the CFD systems in industrial application environments are very similar to characteristics that are required in environments in which CFD software for application in industrial engineering is being developed. For this reason the Informatics Division of NLR implemented quality management for mathematical modelling and software engineering and production. This was certified according to ISO 9001 industry standard requirements, and AQAP-110 military standard requirements. It was applied to the development of a RANS solver to define how quality as defined above can be applied in CFD software [8].

Organizational aspects of cooperative development and quality control of software such as for CFD are described in [4]. A number of technical means to realize it in accordance with the organizational approach have to be supplied by the CSCW environment. This concerns tools to find information that is only required at a few occasions and that can have been changed since the last time that the information was used. It also concerns tools that avoid that the user of the environment does have to deal with questions such as: on which computer might the information have been stored, how do I transfer the information to the system I am working on and how do I find the data set that was used to verify the last version of this software.

The first thing that has to be realised to avoid problems for the users is that the full heterogeneous computer environment can be approached and applied on the basis of a look-and-feel principle as if it is only one single computer. This concerns the user-interaction, processing, information handling and control for execution of functions by members of specific groups of cooperating specialists. Realizing this will be called functional integration (of a heterogeneous computer and software environment).

In addition to functional integration it is required that the user can identify software versions, reconstruct old versions of software under development, identify data and identify document versions. The CSCW environment has to contain tools to make this possible. Also the individuals who make use of the environment have to apply the tools in a prescribed way that is in accordance with these requirements, in case they add information in any form to the environment. Fulfilling these requirements is a prerequisite to be able to work according to the ISO 9001 standard in development and application of CFD software.

In the recent past functional integration of computer networks and software collections has become more feasible as a result of emerging standards in computer systems and software engineering. As a result of this it has been possible to realise a cost effective

functional integration for CFD at NLR. It concerns ISNaS: Information System for flow simulation based on the Navier Stokes equations.The development of the first version of ISNaS [9] was partly subsidized by the Netherlands Ministries of Education and Sciences and of Transport and Public Works. The developed ISNaS environment was used to support the cooperation of two technological institutes and two universities in the development of tools for simulation of compressible flows around aircraft configurations and incompressible flows in waterworks. The concerning project was finalized after six years in 1992. After that at NLR the results of the project were developed further, making use of tools available in the public domain [10]. This has been done in interaction with the development of a quality management system for engineering and design of information systems, and with the development of architecture and construction principles for software for simulation purposes [8].

ISNaS is implemented on the NLR computer network. This is a UNIX network. The network includes, apart from specific network hardware:

- a NEC SX-3 supercomputer
- a NEC CENJU-3 distributed-memory parallel computer
- three CD 4680 mainframes (supplied by Control Data)
- a UP4800 server (supplied by NEC)
- a StorageTek ACS tape robot
- 100 Silicon Graphics workstations, including a few multi processor Power Challenges
- 100 HP workstations
- a cluster of IBM workstations
- PCs, X terminals, and ASCII terminals.

All computers run a vendor-supplied UNIX.

ISNaS contains [10]:

- A user shell that supports presentation integration in that it provides uniformity with respect to the presentation and manipulation of files and the invocation and execution of programs,
- a tool for managing the versions of the source code of a software product,
- a tool for managing data files in a CSCW environment,
- a tool for managing on-line documents,
- a tool that supports automatic localising software in a network,
- a tool that provides a basis for organising and applying conversion programs, ISNaS is portable. It also has proved to be applicable to integrate workstations from customer organisations in NLR's IT infrastructure, to facilitate remote application of NLR's computers and CFD software. The application of ISNaS for remote access to NLR's computers is supported by national and international investments in the "electronic highway". The two NLR establishments are connected via an 34 Mb/s ATM connection. A 34 Mb/s national research network is under development in the Netherlands and in Europe.

4. CONCLUDING REMARKS

At present the multi-vector computer with shared memory architecture in combination with workstations is the best platform for development and application of aircraft CFD.

NLR has a centralised automation strategy and was able to buy a 32 processor NEC

SX-4 vector computer for R&D in CFD. The computer has a peak performance of 64GFLOP/s.

Because most industries prefer workstation based automation, at NLR the CFD codes also are ported to combinations of workstations. This however limits the applicability of CFD.

If industry can not install new multi vector supercomputers, remote application should be considered by industry, especially because functional integration of the remote computer with local workstations is possible.

REFERENCES

1. J.J. Dongarra, H.W. Meuer, E. Strohmaier, TOP500 Supercomputer Sites June 1995; retrievable from internet by typing: rcp anon@nctlib2.cs.utk.cdu:benchmark/performance performance.
2. P. E. Rubbert, CFD and the Changing World of Airplane Design; AIAA Wright Brothers Lecture, Anaheim California; September 18-23, 1994.
3. C.J. Borland, J.R. Benton, P.D. Frank, T.J. Kao, R.A. Mastro, J-F.M. Barthelemy, Multidisciplinary design optimization of a Commercial Aircraft Wing - An Exploratory Study. 5th Symposium on Multidisciplinary Analysis and Optimization, Panama City Beach, Sept. 7-9, 1994. AIAA-94-4305-CP, pp 505-519.
4. W. Loeve, Engineering of systems for application of scientific computing in industry; published in; G. Halevi and R. Weill (eds), Manufacturing in the era of concurrent engineering 1992; Elsevier Science Publishers B.V. ISBN 0 444 89845 X.
5. J. Lewis (Boeing), R. Cosner (McDonnell Aircraft), C. Fischberg (Pratt& Whitney), presentations for panel discussion, published in: S. Taylor, A. Ecer, J. Peraux,, N. Saofuka (eds), Proceedings of parallel CFD'95, Pasadena, CA, USA, 26-29 June 1995; Elsevier Science B.V..
6. M.E.S. Vogels, A model for performance of a block-structured Navier-Stokes solver on clusters of workstations, published in: S. Taylor, A. Ecer, J. Peraux,, N. Satofuka (eds), Proceedings of parallel CFD'95, Pasadena, CA, USA, 26-29 June 1995. Elsevier Science B.V.
7. M.E.S. Vogels, H. van der Ven, G.J. Hameetman, The need for supercomputers in aerospace research and industry; published in: B. Hertzberger, G. Serazzi (eds), High-Performance Computing and Networking. 1995; Springer-Verlag ISBN 3-540-59393-4.
8. M.E.S. Vogels, "Principles for Software Management of Mathematical Software", 1994 European Simulation Symposium, 9-12 October 1994, Istanbul, Turkey. Also available as NLR report TP 94324.
9. M.E.S. Vogels and W. Loeve, Development of ISNaS: An Information System for Flow Simulation in Design, published in: F. Kimura and A. Rolstadas (eds), Computer Applications in Production and Engineering. 1989; North-Holland ISBN 0 444 88089 5.
10. E.H. Baalbergen and W. Loeve, Spine: software platform for computer supported cooperative work in heterogeneous computer and software environments, NLR TP 94359, 1994; National Aerospace Laboratory Amsterdam.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel computations of unsteady supersonic cavity flows

Yoko TAKAKURA[a], Fumio HIGASHINO[a], Takeshi YOSHIZAWA[a],
Masahiro YOSHIDA[b], and Satoru OGAWA[b]

[a] Department of Mechanical Systems Engineering, Tokyo Noko University, 2-24-16 Nakamachi, Koganei-shi, Tokyo 184, Japan

[b] Computational Sciences Division, National Aerospace Laboratory, 7-44-1 Jindaiji-Higashi-machi, Chofu-shi, Tokyo 182, Japan

Vector-Parallel computations of unsteady supersonic cavity flows have been executed on the NWT system using a very large eddy simulation. In 2-D computations on rough grids the characteristic patterns of vortex structures have been recognized according to the ratio of length to depth of cavity, accompanied with the low-frequent oscillation for the upward and downward motion of a shear layer and for the appearance and disappearance of shock waves. It is caused by the inflow and outflow of fluid for the cavity. The effect of an incident oblique shock wave is to make a right-handed vortex grow through the inflow of mean flow into the cavity. 2-D computation using a fine grid has indicated another oscillation with higher frequency caused by the generation of a small vortex in the shear layer itself and its disappearance with the collision against the cavity wall. These two frequencies agree with the experiments. In 3-D computation, three-dimensional nature of shear layer has been recognized. 3-D computation has shown the better parallel performance than 2-D one, and NWT is hopeful for solving the fluid physics through large-scale computations.

1. INTRODUCTION

In the case of supersonic flows within a duct having a rectangular cavity, it is reported experimentally[1,2] that characteristic structure of vortices is recognized within a cavity according to the ratio of length to depth of cavity, and that it is accompanied by the oscillation of a shear layer between the mean flow in the duct and the vortical flow within the cavity. Further when an oblique shock wave is made incident to the shear layer, an interaction occurs among the vortices, the shear layer and the incident shock wave. This interacting phenomenon is significantly interesting from the viewpoint of fluid dynamics, as well as from the viewpoint of engineering application to flow control for the supersonic intake of a RAM-jet engine, etc..

The aim of the present study is to solve the physical mechanics of the unsteady supersonic cavity flows where the above interaction occurs by large-scale computations, focusing on the vortex structure, the origins of oscillations, and the effect of an incident oblique shock wave.

The numerical method used is a very large eddy simulation[3,4] to capture the unsteady oscillation, and the computing code is implemented for the numerical wind tunnel (NWT) system in National Aerospace Laboratory (NAL).

2. METHOD OF NUMERICAL ANALYSIS

2.1. Discretization

A very large eddy simulation[3] is carried out, where the governing equations are the filtered 2-D and 3-D Navier-Stokes equations together with the sub-grid scale eddy viscosity written in [4]. The finite volume method is adopted for discretization: spatial discretization is executed by the Chakravarthy-Osher TVD scheme[5] improved in linearization[6] and time integration is carried out by the second-order Runge-Kutta scheme[7].

2.2. Parallel implementation

The computing code is implemented for the NWT system[8], a parallel computer system with distributed memory. It is composed of 140 processor elements (PEs) which is combined under a cross-bar interconnection network. Each PE is a vector computer with 1.7 GFLOPS peak performance and 256 MB main memory. Figure 1 shows the configuration of the NWT system.

In the vector-parallel implementation two arrays are provided for the usual one data set. In the 3-D case one array is partitioned in the z-direction with each partitioned data stored in the individual PEs. This array is used when computing the data vectorized for the x- and y-direction in parallel . Similarly the other array partitioned in the x- or y-direction is used to compute the data vectorized for the remaining direction, z. When the algorithm needs to change the direction of vectors, data are transferred from one array to the other by the transpose move through the cross-bar network.

Figure 2 shows an example of the transpose move from the z-partitioned array to the y-partitioned array in the case of 4 PEs. As is indicated by arrows, for example, the data stored in PE 2 within the z-partitioned array are transferred to all PEs. This transpose move, that is to say, all to all move using the cross-bar network, makes the efficient vector-parallel algorithm executable.

3. COMPUTATIONAL CONDITIONS

Figure 3 shows the schematic view of a supersonic duct having a rectangular cavity and a shock generator. The basic dimension is given by height of duct (H), depth of cavity (D), and length of cavity (L). These sizes are determined identical with those in the experiments of our laboratory[2]: $H = 2cm, D = 1cm$, and $L/D = 1, 2$ and 3. Mach number of mean flow is given by $M_\infty = 1.83$.

First, in order to obtain the overview of the flow stated above, 2-D cavity flows are numerically solved on rough grids by adopting L/D as a parameter without or with the incident oblique shock wave created by the shock generator. Second, to investigate the effect of the shear layer in more detail, a fine grid is used. Finally, to investigate the three-dimensional nature of the flow-field, the 3-D cavity flows are computed.

4. RESULTS AND DISCUSSION

4.1. 2-D cavity flows on rough grids

Here 2-D cavity flows are numerically solved on rough grids with number of grid points, 453×153, in the cases of $L/D = 1, 2$ and 3 without or with the shock generator.

a) Unsteadiness: $L/D = 2$ without a shock generator Figure 4 shows the time-evolving solution with density contours and instantaneous stream lines. At time stage (a) the two vortices, the distinct vortex having the right-handed rotation in the downstream and the indistinct vortex in the upstream, are recognized within the cavity. At (b) the

outflow from the cavity occurs, accompanied with the growth of the downstream vortex. It is also observed that the shape of the bow shock wave at the rear corner of the cavity has abruptly changed during (a) and (b). At (c) when the outflow continues, the pressure and density within the cavity become low. Further when they become lower by the outflow, at time stage (d) the fluid starts to flow into the cavity and the upstream vortex with the left-handed rotation becomes clear. During (c) and (d), since the vortical flow within the cavity rises upward, it is observed that the oblique shock wave appears from the front corner of the cavity. At (e) when the pressure is recovered by the inflow, not only the inflow but also the outflow occurs simultaneously. After this state, the state (a) comes again, where the upstream vortex is indistinct, the front oblique shock wave disappears, and the rear bow shock wave is confused. Thus it has been recognized that the inflow and outflow causes the oscillation such as rise and fall of the boundary surface (shear layer) between the duct flow and the cavity flow, through the growth and decline of vortices. Accompanied with this motion, the oscillation of the shock waves, i.e., the oblique shock wave from the front corner of the cavity and the bow shock wave at the rear corner of the cavity, also occurs. Since no boundary condition forcing the time-dependency is imposed, a series of phenomena is considered as a self-sustained oscillation.

The period time for this oscillation is about $50 \sim 60\mu sec.$, which agrees with experimental one in our laboratory[2], and its time scale, T, is represented by $T \sim L/U_\infty$ (U_∞: mean flow velocity). Also from this fact it is considered that this oscillation is mainly caused by the inflow and outflow of mean flow for the cavity.

b) Effect of an incident oblique shock wave: $L/D = 2$ with a shock generator Figure 5 (a) and (b) show respectively the density contours and stream lines obtained by averaging in time the unsteady solution with a shock generator. Compared with Fig.4, it is observed that the upstream vortex almost vanishes and instead the downstream vortex with the right-handed rotation is developed. The reason is, we consider, that the oblique shock wave changes the direction of mean flow downward, and consequently the inflow into the cavity is caused in the downstream of the shock wave, so that the right-handed vortex might grow. Comparison between Fig.5 (a) and (c) indicates that the time-averaged solution in computation agrees well with the schlieren photography in experiments. Referring to the unsteadiness, the period time for the oscillation is about same in both the cases without and with the shock generator.

c) Vortex structures: $L/D = 1, 2$ and 3 without or with a shock generator Figure 6 (a) and (b) show solutions in the case of $L/D = 1$ without and with a shock generator, respectively. In the instantaneous solution without an incident shock wave (Fig.6 (a)) one can find a right-handed vortex within the cavity. When the shock generator is installed, the time-averaged solution (Fig.6 (b)) shows that vortex becomes remarkable because of the inflow effect of the incident oblique shock wave.

Similarly Figure 7 (a) and (b) show instantaneous solutions in the case of $L/D = 3$ without and with the shock generator, respectively. In Fig.7 (a) without an incident shock, mainly two vortices are observed. But evolving the time, another vortex appears in the upstream of the cavity: totally there are three vortices. Regarding the directions of vortices, see Fig.8. When the shock generator is installed in the position of Fig.7 (b), the upstream right-handed vortex, over which the oblique shock wave is incident, grows.

The vortex patterns in the cavities are summarized in Fig.8. For the cavities with $L/D = 1, 2$ and 3, one, two and three vortices appear, respectively. First in the downstream of the cavity a vortex with right-handed rotation is created (for $L/D = 1, 2$ and 3). Induced by the downstream vortex, the second vortex with left-handed rotation is generated (for $L/D = 2$ and 3). Further induced by the second vortex, the third vortex with right-handed rotation is generated (for $L/D = 3$).

4.2. 2-D cavity flows on a fine grid

Here 2-D cavity flows are numerically solved in the case of $L/D = 2$ with a shock generator, using the fine grid (Fig.9) with number of points 927×479 where points are gathered near the shear layer.

Figure 10 shows the instantaneous density contours. One can see a downstream vortex in the bottom of cavity, and additionally a small vortex in the shear layer. In the time-evolving solution, the vortex within the cavity moves with the right-handed rotation. On the other hand, the small vortex generated in the shear layer is floated downstream, and disappears with the collision against the cavity wall. The period time for the former, that is, motion of vortex within the cavity, is nearly same as that on the rough grid . The period time for the latter, that is, motion of the vortex in the shear layer, however, has the different value: about half of the time for the former motion in this case.

Thus two oscillations have been recognized: one oscillation with low frequency is caused by the inflow and outflow for the cavity, which was shown in Sec.4.1, and the other with higher frequency is caused by the shear layer itself. These frequencies are also detected in the experiments[2].

4.3. 3-D cavity flows

Here 3-D cavity flows are numerically solved in the case of $L/D = 2$ without a shock generator, with number of grid points $201 \times 171 \times 63$. In this case the width of the computational region is taken as the same size as the cavity length, and the periodic boundary condition is imposed in the spanwise direction.

Figure 11 shows the instantaneous density contours on the sections at the middle of span, at the middle of cavity, and at the bottom of duct. In this figure two period of waves in the spanwise direction are recognized in the shear layer. However, further investigation would be needed to understand the three-dimensional feature in the computation.

4.4. Parallel performance

Figure 12 shows the parallel performance, where number of grid points are limited so that computation could be executed even on a PE . As shown in Fig.12 (a), in 2-D case the speed-up ratio begins to be saturated at 16 PEs, and the efficiency for 16 PEs is about 70%. On the contrary in 3-D case shown in Fig.12 (b), the speed-up ratio is not inclined to be saturated, and the efficiency for 52 PEs is about 75%. The reason why the 3-D case indicates the better performance than 2-D one would be that the time ratio of the transpose move to the total computation decreases in 3-D case.

5. CONCLUDING REMARKS

Vector-Parallel computations of unsteady supersonic cavity flows have been executed using the NWT system.

In 2-D computations on rough grids, one, two and three vortices have been recognized for the cavity of $L/D = 1, 2$ and 3 respectively with the vortex structures of Fig.8, accompanied with the oscillation for the upward and downward motion of the shear layer and for the appearance and disappearance of the front oblique shock wave and the rear bow shock wave. This oscillation is caused mainly by the inflow and outflow of fluid for the cavity. The effect of an oblique shock wave which is made incident over the shear layer is to make a right-handed vortex grow through the inflow of mean flow into the cavity.

2-D computation using a fine grid has indicated another oscillation with higher frequency caused by the generation of a small vortex in the shear layer itself and its disappearance with the collision against the cavity wall, together with the low-frequent oscillation caused by the inflow and outflow. These two frequencies agree with the experiments.

In 3-D computation, three-dimensional nature of shear layer has been recognized, but further investigation would be needed to understand the computational results.

Referring to the parallel performance, 3-D computation is better than 2-D one for the vector-parallel implementation using the transpose move, and therefore NWT is hopeful for solving the fluid physics through large-scale computations.

REFERENCES

1. X. Zhang and J.A. Edwards, An Investigation of Supersonic Oscillatory Cavity Flows Driven by Thick Shear Layers, Aeronautical J. December (1990) 355.
2. Y. Watanabe, S. Kudo and F.Higashino, Effects of Cavity Configurations on the Supersonic Internal Flow, International Symposium on Shock Waves, July (1995).
3. D. Hunt, A Very Large Eddy Simulation of an Unsteady Shock Wave/Turbulent Boundary Layer Interaction, AIAA Paper 95-2212 (1995).
4. Y. Takakura, S. Ogawa and T. Ishiguro, Turbulence Models for 3-D transonic Viscous Flows, AIAA Paper 89-1952-CP (1989).
5. S.R. Chakravarthy and S. Osher, A New Class of High Accuracy TVD Schemes for Hyperbolic Conservation Laws, AIAA Paper 85-0363 (1985).
6. Y. Takakura, T. Ishiguro and S. Ogawa, On TVD Difference Schemes for the Three-Dimensional Euler Equations in General Co-Ordinates, Int. J. Numerical Methods in Fluids, Vol.9 (1989) 1011.
7. A. Rizzi, Damped Euler-Equation Method to Compute Transonic Flow Around Wing-Body Combinations, AIAA J. Vol.20 No.10 (1982) 1321.
8. H. Miyoshi, Y. Yoshioka, M. Ikeda and M. Takamura, Numerical Wind Tunnel Hardware, NAL SP-16 (1991) 99.

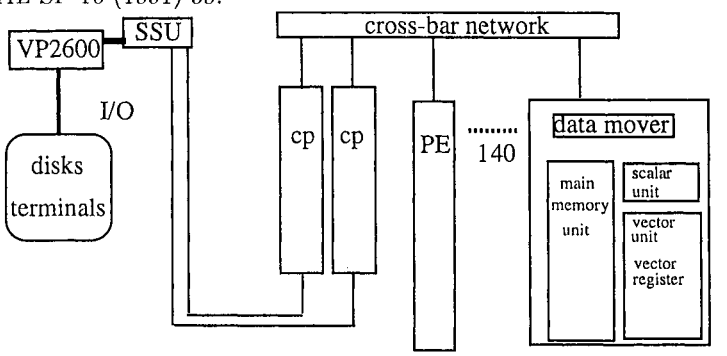


Figure 1. Configuration of NWT system.

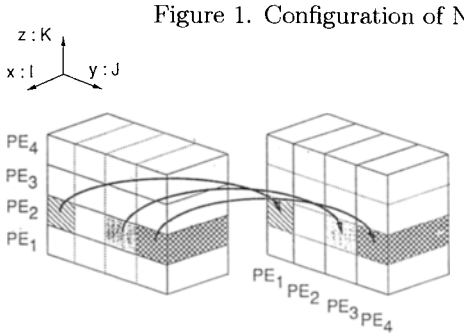


Figure 2. Transpose move in case of 4 PEs.

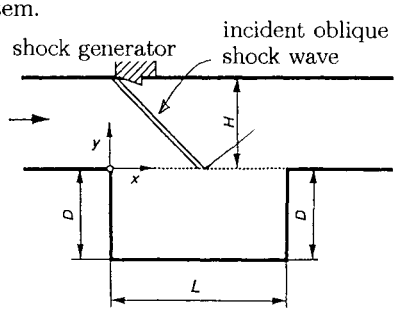


Figure 3. Schematic view of duct having cavity and shock generator.

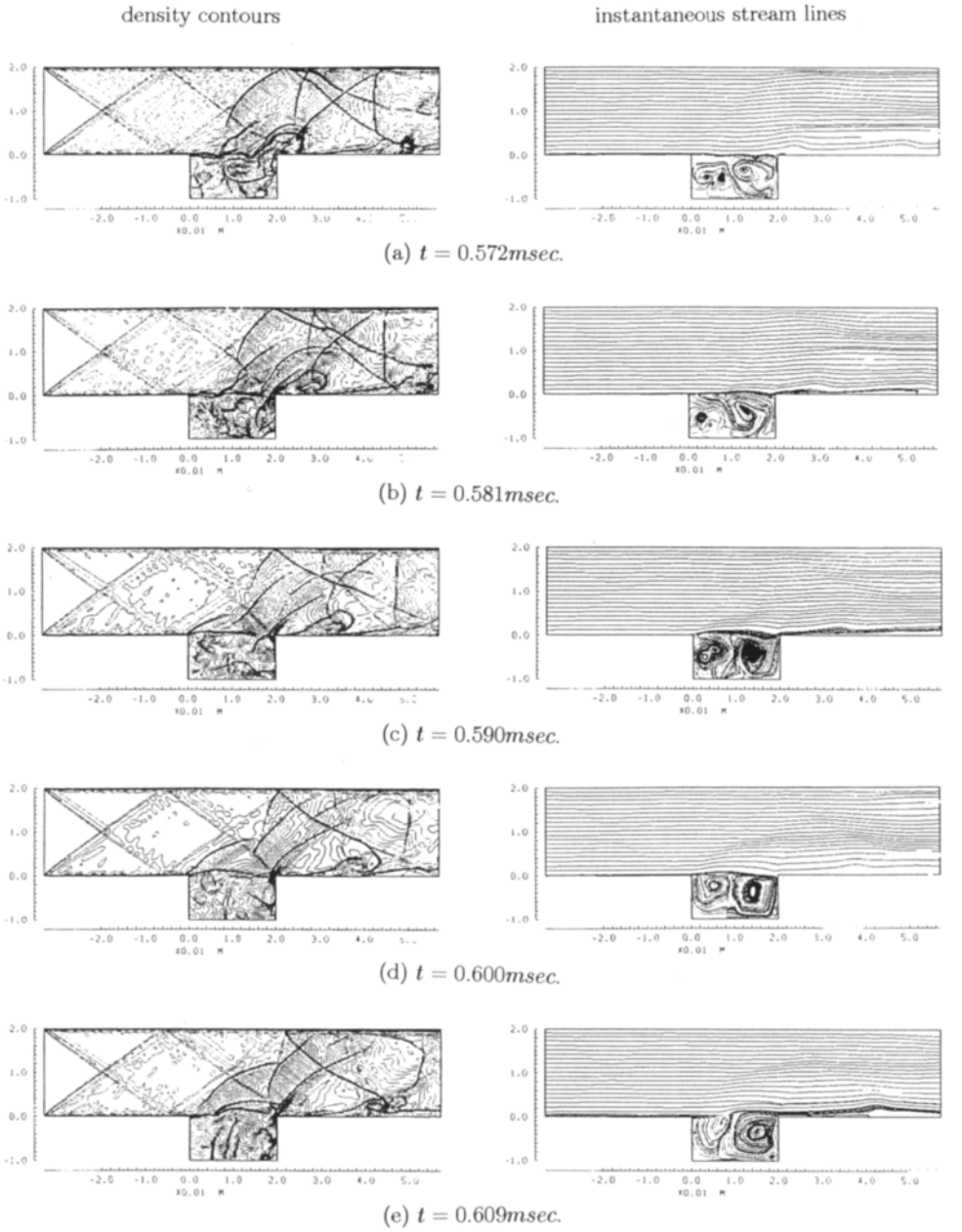
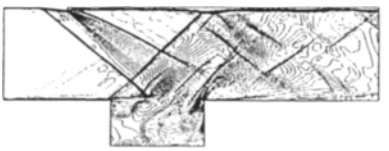
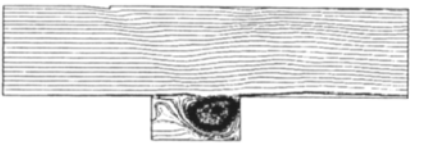


Figure 4. Time-evolutional solution in case of $L/D = 2$ without shock generator: density contours and stream lines.



(a) Density contours.



(b) Stream lines.

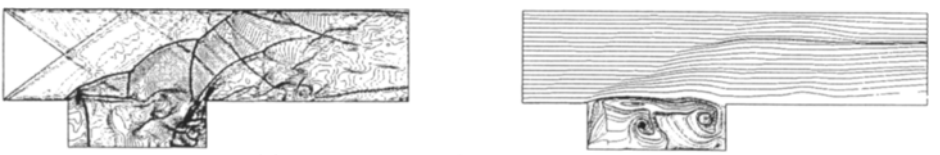
(c) Schlieren Photography in experiments.

Figure 5. Time-averaged solution in case of $L/D = 2$ with shock generator.

(a) Instantaneous solution without shock generator.

(b) Time-averaged solution with shock generator.

Figure 6. Solutions in case of $L/D = 1$: density contours and stream lines.



(a) Instantaneous solution without shock generator.

(b) Instantaneous solution with shock generator.

Figure 7. Solutions in case of $L/D = 3$ density contours and stream lines.

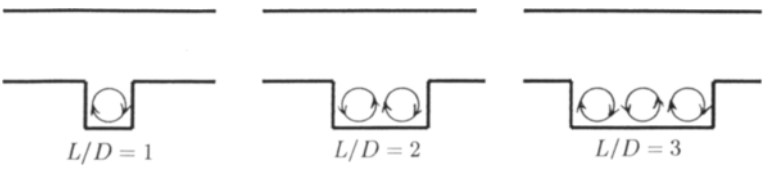


Figure 8. Characteristic patterns of vortex structures.

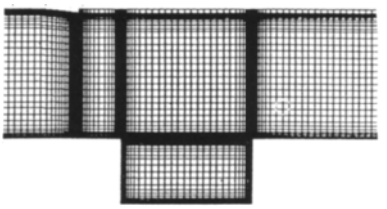


Figure 9. 2-D fine grid.

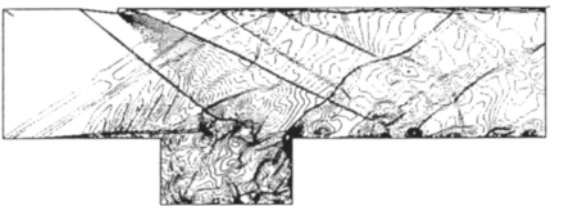


Figure 10. Instantaneous density contours on fine grid in case of $L/D = 2$ with shock generator.

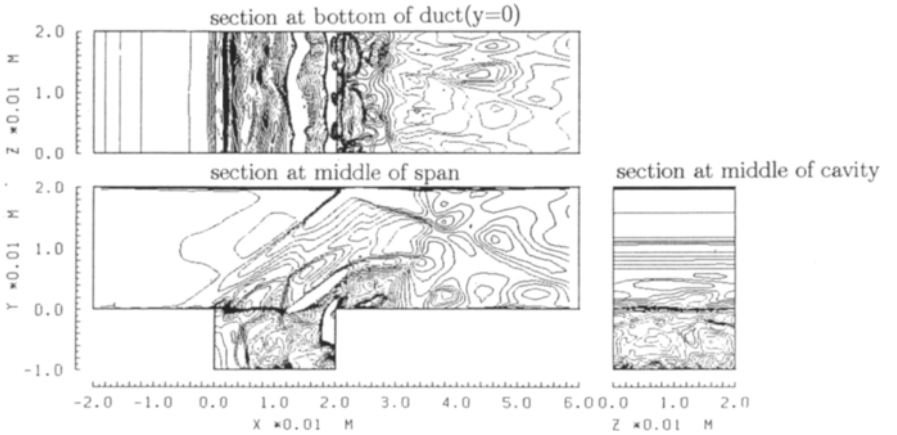
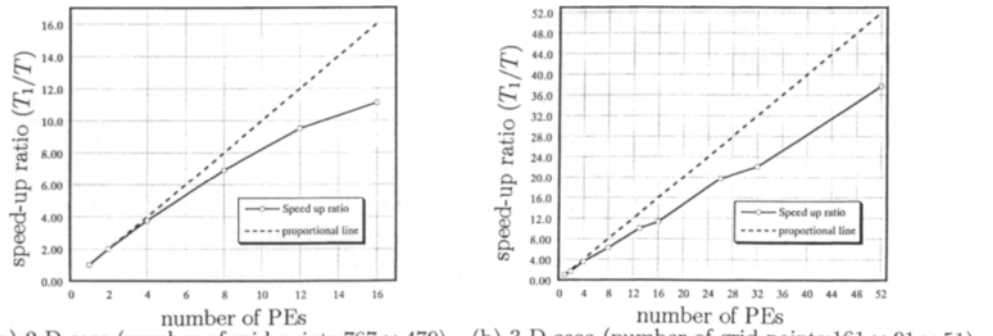


Figure 11. 3-D solution in case of $L/D = 2$ without shock generator: density contours.



(a) 2-D case (number of grid points:767 × 479). (b) 3-D case (number of grid points:161 × 91 × 51).

Figure 12. Parallel performance: speed up ratio.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A Data-Parallel LU Relaxation Method for Reacting Viscous Flows

M. J. Wright and G. V. Candler*

Department of Aerospace Engineering & Mechanics and Army High Performance Computing Research Center, University of Minnesota, Minneapolis MN 55455, USA

The LU-SGS method of Yoon and Jameson is modified for the simulation of reacting viscous flows on massively parallel computers. The resulting data-parallel lower-upper relaxation method (DP-LUR) is shown to have good convergence properties for many problems. The new method is implemented on the Thinking Machines CM-5, where a large fraction of the peak theoretical performance of the machine is obtained. The low memory use and high parallel efficiency of the method make it attractive for large-scale simulation of reacting viscous flows.

1. INTRODUCTION

The large cost associated with solving the reacting Navier-Stokes equations makes the use of a massively parallel supercomputer (MPP) attractive, since such machines have a very large peak performance. However, it is difficult to efficiently implement most implicit methods on an MPP, since a true implicit method requires the inversion of a large sparse matrix, which involves a great deal of inter-processor communication. The traditional solution to this problem is to perform some sort of domain decomposition, which reduces the amount of communication required by solving the implicit problem on a series of local sub-domains. This approach has been used with good success, but the resulting algorithms can become costly and complicated due to load balancing and boundary update issues.[1]

Another approach is to seek an implicit method which would be amenable to data-parallel implementation without domain decomposition. Such an algorithm would then be readily portable to a wide variety of parallel architectures, since it is relatively easy to run a data-parallel code on a message-passing machine. The Lower-Upper Symmetric Gauss-Seidel (LU-SGS) method of Yoon and Jameson[2] is a good starting point, because it makes some approximations to the implicit problem which eliminate the need for large block matrix inversions. Candler, Wright, and McDonald have shown[3] that it is possible to modify the LU-SGS method, making it almost perfectly data-parallel for inviscid flows. The resulting diagonal data-parallel lower-upper relaxation (DP-LUR) method replaces

* Authors supported by the NASA Langley Research Center Contract NAG-1-1498. This work is also sponsored in part by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

the Gauss-Seidel sweeps of the LU-SGS method with a series of pointwise relaxation steps. With this method all data dependencies are removed, and the computation of each relaxation step becomes perfectly data-parallel. Also, the formulation of the method ensures proper load balancing at all times, eliminating the need for a domain decomposition. The diagonal method has been further modified by Wright, Candler, and Prampolini[4] to improve the performance for viscous flow simulations, and the resulting full matrix DP-LUR method retains all of the excellent parallel attributes of the original.

This paper presents the evolution of the diagonal and full matrix data-parallel LU relaxation methods, and discusses their implementation on the Thinking Machines CM-5. The convergence rates of the methods on several test cases are examined, and their parallel performance on the CM-5 is presented.

2. DIAGONAL DP-LUR

The fully implicit form of the two-dimensional Navier-Stokes equations is

$$\frac{U^{n+1}-U^n}{\Delta t}+\frac{\partial \tilde{F}^{n+1}}{\partial \xi}+\frac{\partial \tilde{G}^{n+1}}{\partial \eta}=W^{n+1},$$

where U is the vector of conserved quantities, $\tilde{F}$ and $\tilde{G}$ are the flux vectors in the ξ (body-tangential) and η (body-normal) directions, and W is the source vector for chemical reactions and internal energy relaxation. The flux vectors can be split into convective and viscous parts

$$\tilde{F}=F+F_v, \qquad \tilde{G}=G+G_v.$$

If we focus on the inviscid problem for now, we can linearize the flux vector using

$$F^{n+1}\simeq F^n+\left(\frac{\partial F}{\partial U}\right)^n(U^{n+1}-U^n)=F^n+A^n\delta U^n,$$
$$G^{n+1}\simeq G^n+B^n\delta U^n, \qquad W^{n+1}\simeq W^n+C^n\delta U^n.$$

We then split the fluxes according to the sign of the eigenvalues of the Jacobians to obtain the upwind finite volume representation

$$\begin{aligned}\delta U^n_{i,j}+\frac{\Delta t}{V_{i,j}}\Big\{&\left(A_{+i+\frac{1}{2},j}S_{i+\frac{1}{2},j}\delta U_{i,j}-A_{+i-\frac{1}{2},j}S_{i-\frac{1}{2},j}\delta U_{i-1,j}\right)\\&+\left(A_{-i+\frac{1}{2},j}S_{i+\frac{1}{2},j}\delta U_{i+1,j}-A_{-i-\frac{1}{2},j}S_{i-\frac{1}{2},j}\delta U_{i,j}\right)\\&+\left(B_{+i,j+\frac{1}{2}}S_{i,j+\frac{1}{2}}\delta U_{i,j}-B_{+i,j-\frac{1}{2}}S_{i,j-\frac{1}{2}}\delta U_{i,j-1}\right)\\&+\left(B_{-i,j+\frac{1}{2}}S_{i,j+\frac{1}{2}}\delta U_{i,j+1}-B_{-i,j-\frac{1}{2}}S_{i,j-\frac{1}{2}}\delta U_{i,j}\right)\Big\}^n-\Delta t\,C^n_{i,j}\delta U^n_{i,j}=\Delta t\,R^n_{i,j},\end{aligned} \tag{1}$$

where $R^n_{i,j}$ is the change of the solution due to the fluxes at time level n, S is the surface area of the cell face indicated by its indices, and $V_{i,j}$ is the cell volume.

The LU-SGS method of Yoon and Jameson[2] is a logical choice for implicit time advancement, because it makes some simplifications to the implicit equation which diagonalize the problem and allow steady-state solutions to be obtained with a dramatically

reduced number of iterations over an explicit method, without a substantial increase in the computational cost per iteration. In addition, the extension to three-dimensional flows is straightforward. Although the method as formulated does not lend itself to implementation on a parallel machine, we will see that it is possible to make some modifications to the method that make it perfectly data-parallel. Following the method of Yoon and Jameson, we approximate the implicit Jacobians in such a manner that differences between the Jacobians become, for example,

$$A_+ - A_- = \rho_A I,$$

where ρ_A is the spectral radius of the Jacobian A, given by the magnitude of the largest eigenvalue $|u| + a$, where a is the speed of sound. If we then move the off-diagonal terms to the right-hand side, the resulting implicit equation becomes

$$\begin{aligned}
&\left\{I + \lambda_A I + \lambda_B I + \Delta t \,\mathrm{diag}(C)^n_{i,j}\right\}^n_{i,j} \delta U^n_{i,j} = \Delta t\, R^n_{i,j} \\
&+\frac{\Delta t}{V_{i,j}} A^n_{+i-\frac{1}{2},j} S_{i-\frac{1}{2},j} \delta U^n_{i-1,j} - \frac{\Delta t}{V_{i,j}} A^n_{-i+\frac{1}{2},j} S_{i+\frac{1}{2},j} \delta U^n_{i+1,j} \\
&+\frac{\Delta t}{V_{i,j}} B^n_{+i,j-\frac{1}{2}} S_{i,j-\frac{1}{2}} \delta U^n_{i,j-1} - \frac{\Delta t}{V_{i,j}} B^n_{-i,j+\frac{1}{2}} S_{i,j+\frac{1}{2}} \delta U^n_{i,j+1},
\end{aligned} \tag{2}$$

where $\lambda_A = \frac{\Delta t S}{V} \rho_A$. An appropriate diagonalized form for the source term Jacobian C is given by Hassan et al.[5] The viscous fluxes can be linearized in a manner similar to the Euler terms following the work of Tysinger and Caughey,[6] or Gnoffo,[7] and can be included easily in (2) without changing the basic nature of the method.[4] With these approximations, (2) can be solved without any costly matrix inversions. It is important to note that we are making approximations only to the implicit side of the problem; therefore the steady-state solution will be unaffected.

The LU-SGS algorithm employs a series of corner-to-corner sweeps through the flowfield using the latest available data for the off-diagonal terms to solve (2). This method has been shown to be efficient on a serial or vector machine. However, significant modifications are required to reduce or eliminate the data dependencies that are inherent in (2) and make the method parallelize effectively. The data-parallel LU relaxation (DP-LUR) approach solves this problem by replacing the Gauss-Seidel sweeps with a series of pointwise relaxation steps using the following scheme. First, the off-diagonal terms are neglected and the right-hand side, $R_{i,j}$, is divided by the diagonal operator to obtain $\delta U^{(0)}$

$$\delta U^{(0)}_{i,j} = \left\{I + \lambda^n_A I + \lambda^n_B I + \Delta t \,\mathrm{diag}(C)^n_{i,j}\right\}^{-1}_{i,j} \Delta t\, R^n_{i,j}.$$

Then a series of k_{max} relaxation steps are made using

for $k = 1,\ k_{max}$

$$\begin{aligned}
\delta U^{(k)}_{i,j} = &\left\{I + \lambda^n_B I + \lambda^n_B I + \Delta t \,\mathrm{diag}(C)^n_{i,j}\right\}^{-1}_{i,j} \Big\{\Delta t\, R^n_{i,j} \\
&+ \frac{\Delta t}{V_{i,j}} A^n_{+i-\frac{1}{2},j} S_{i-\frac{1}{2},j} \delta U^{(k-1)}_{i-1,j} - \frac{\Delta t}{V_{i,j}} A^n_{-i+\frac{1}{2},j} S_{i+\frac{1}{2},j} \delta U^{(k-1)}_{i+1,j} \\
&+ \frac{\Delta t}{V_{i,j}} B^n_{+i,j-\frac{1}{2}} S_{i,j-\frac{1}{2}} \delta U^{(k-1)}_{i,j-1} - \frac{\Delta t}{V_{i,j}} B^n_{-i,j+\frac{1}{2}} S_{i,j+\frac{1}{2}} \delta U^{(k-1)}_{i,j+1} \Big\}
\end{aligned} \tag{3}$$

then

$$\delta U^{n+1}_{i,j} = \delta U^{(k_{max})}_{i,j}.$$

With this approach, all data required for each relaxation step have already been computed during the previous step. Therefore, the entire relaxation step is performed simultaneously in parallel without any data dependencies, and all communication can be handled by optimized nearest-neighbor routines. In addition, since the same pointwise calculation is performed on each computational cell, load balancing will be ensured as long as the data are evenly distributed across the available processors. The low memory usage of the method makes is possible to solve up to 32M grid points on a 512 processor CM-5. These facts, together with the nearly perfect scalability of the algorithm to any number of processing nodes, make it attractive for the solution of very large problems.

The diagonal DP-LUR method has been tested on two-dimensional geometries, with an emphasis on evaluating the convergence properties and parallel performance of the method. The primary test case for this paper is the Mach 15 flow over a cylinder-wedge body, with Reynolds numbers based on freestream conditions varying from 3000 to 30×10^6. The boundary layer resolution is measured with the wall variable $y_+ = \rho y u_* / \mu$, where u_* is the friction velocity. For a well resolved boundary layer the mesh spacing is typically chosen so that $y_+ \leq 1$ for the first cell above the body surface. The grid is then exponentially stretched from the wall to the outer boundary, which results in approximately 70 points in the boundary layer for the baseline 128 × 128 grid. Although the results presented here are based on modified first order Steger-Warming flux vector splitting for the explicit fluxes,[8] it is important to note that the derivation of the implicit algorithm is general and can be used with many explicit methods. Results have also been obtained using a Harten-Yee upwind non-MUSCL TVD scheme,[9] with comparable convergence properties.

The effect of the number of relaxation steps (k_{max}) on convergence is shown in Fig. 1a for perfect gas flow over a cylinder-wedge at $Re = 30 \times 10^3$ and $y_+ = 1$. We see that the convergence rate steadily improves as k_{max} increases, to a maximum of $k_{max} = 4$. It is possible to further increase the number of relaxation steps, but only small improvements in convergence are achieved, and the algorithm becomes less stable for many problems. Figure 1b plots the convergence rates for the same case versus computer time on a 64 processor partition of the CM-5. Because each relaxation step has a small cost compared to the evaluation of the explicit fluxes, $k_{max} = 4$ is also the most cost-effective approach.

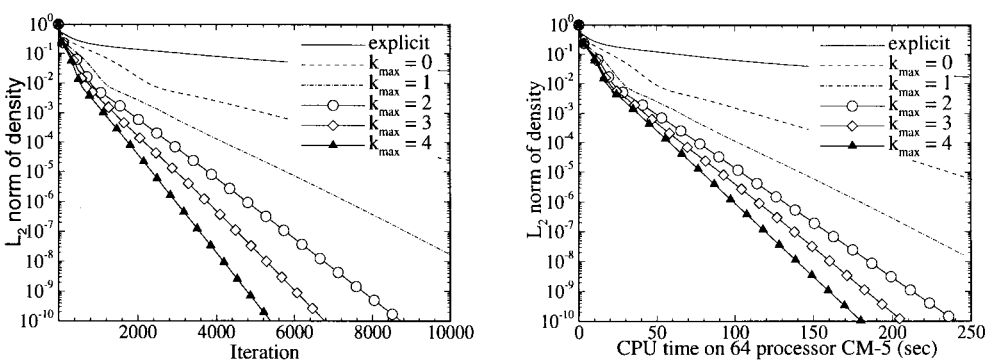


Figure 1. a) Convergence histories, and b) CPU times on 64 processor CM-5, for the perfect gas diagonal DP-LUR method showing influence of k_{max}. Cylinder-wedge body at $M_\infty = 15$ and $Re = 3 \times 10^4$. 128 × 128 grid with $y_+ = 1$.

Figure 2 compares the convergence rate of the new DP-LUR method to the original LU-SGS method. The flow conditions are the same as for Fig. 1. We see that, while the performance of the two methods is similar at $k_{max} = 2$, the DP-LUR method converges in significantly fewer iterations when four relaxation steps ($k_{max} = 4$) are used. This result is surprising, because the relaxation technique limits the distance that information can travel during each time step. However, this trend holds true for all cases tested to date.

The reacting flow version of the DP-LUR method is implemented for a five-species chemical and vibrational nonequilibrium model for air, using standard reaction rates.[10] Figure 3 presents the convergence rate for an inviscid reacting flow case, together with a perfect gas solution on the same grid. The freestream conditions for both are Mach 15 flow at 60 km. We see that the convergence rate is nearly identical for both the perfect gas and non-equilibrium chemistry simulations. This result holds approximately true for many cases, although the reacting flow convergence rate does show some degradation as the stiffness of the problem increases (increasing density or speed).

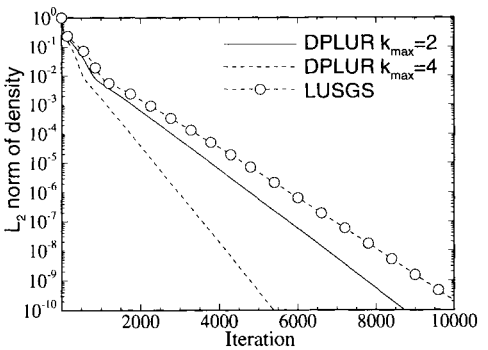


Figure 2. Convergence histories for the DP-LUR and original LU-SGS methods. Cylinder-wedge body at $M_\infty = 15$ and $Re = 30 \times 10^3$. 128×128 grid with $y_+ = 1$.

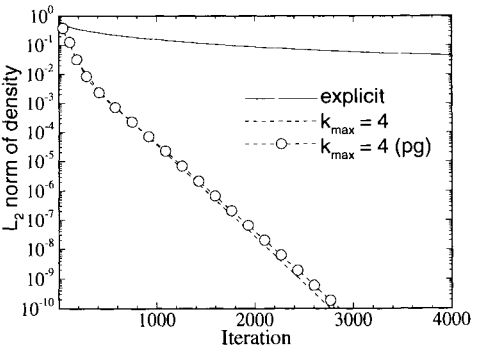


Figure 3. Convergence histories for the inviscid perfect gas and five species reacting air DP-LUR methods. Cylinder-wedge body at $M_\infty = 15$ and 60km, 128×128 grid.

3. FULL MATRIX DP-LUR

While these results show that the diagonal method is very effective for many problems, it is well known that methods of this type exhibit significant degradation of convergence rate on the high cell aspect ratio (CAR) grids necessary for the simulation of high Reynolds number flows.[5,11] One way to alleviate this problem is to remove the Yoon and Jameson diagonalizing approximation and use the exact split Jacobians on and off the diagonal. This results in

$$\begin{aligned}\left\{I + \frac{\Delta t}{V_{i,j}}\left[A_{+i+\frac{1}{2},j}S_{i+\frac{1}{2},j} - A_{-i-\frac{1}{2},j}S_{i-\frac{1}{2},j} + B_{+i,j+\frac{1}{2}}S_{i,j+\frac{1}{2}} - B_{-i,j-\frac{1}{2}}S_{i,j-\frac{1}{2}}\right] - \Delta t\, C_{i,j}\right\}^n \delta U^n_{i,j} \\ = \Delta t\, R^n_{i,j} + \frac{\Delta t}{V_{i,j}}A^n_{+i-\frac{1}{2},j}S_{i-\frac{1}{2},j}\delta U^n_{i-1,j} - \frac{\Delta t}{V_{i,j}}A^n_{-i+\frac{1}{2},j}S_{i+\frac{1}{2},j}\delta U^n_{i+1,j} \\ + \frac{\Delta t}{V_{i,j}}B^n_{+i,j-\frac{1}{2}}S_{i,j-\frac{1}{2}}\delta U^n_{i,j-1} - \frac{\Delta t}{V_{i,j}}B^n_{-i,j+\frac{1}{2}}S_{i,j+\frac{1}{2}}\delta U^n_{i,j+1}.\end{aligned} \tag{4}$$

The full matrix form of the DP-LUR method then follows logically if we use the same approach to solve (4) as shown in (3). The resulting method is somewhat more computationally and memory intensive than the diagonal version, since a $n_{eq} \times n_{eq}$ matrix must be inverted and stored at every grid point, where n_{eq} is the number of equations in the system. However, that calculation requires only local data and must be performed only once per iteration. The overall algorithm remains entirely data-parallel and free of all data dependencies, and in fact requires no more communication per iteration than the diagonal version.

Figure 4a presents the computer time on a 64 processor CM-5 required for the error norm to fall ten orders of magnitude for the two perfect gas inviscid algorithms as a function of the maximum cell aspect ratio. We can see that although the more complex full matrix method takes about 1.7 times as long per iteration, the overall solution time is slightly better than the diagonal method even for the lowest CAR grids. As the CAR increases, both methods require more iterations to reach steady-state, but the full matrix method is less affected by the grid, and converges more than two times faster on the highest CAR tested (CAR = 10,000). In Fig. 4b we also plot computer time for ten orders of error norm reduction versus CAR, but this time for a low Reynolds number ($Re = 3000$) perfect gas flow. For this case the full matrix method is superior on all grids, and in fact is about 20 times faster on the highest aspect ratio grid tested. This shows that the more physical approach of the full matrix method is clearly better suited to the solution of highly viscous (low Re) flows.

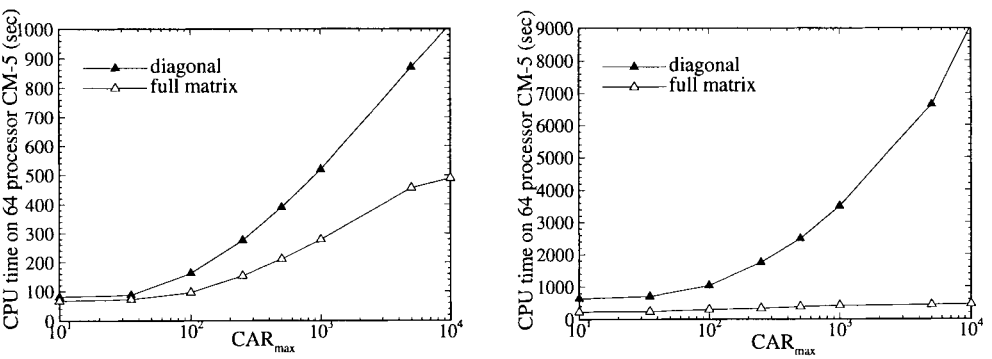


Figure 4. CPU times on 64 processor CM-5 to achieve ten orders of error norm convergence for the full matrix and diagonal perfect gas DP-LUR methods as a function of maximum cell aspect ratio. a) Inviscid, and b) viscous with $Re = 3000$. Cylinder-wedge body at $M_\infty = 15$. 128×128 grid with $y_+ = 1$ for each case.

An important result from Fig. 4 is that, for all 2-D perfect gas cases run to date, the full matrix method achieves convergence faster than the diagonal. However, this does not mean that the diagonal method should be discarded. The 2-D implementation of the diagonal method uses about 30% less memory than the full matrix method; therefore it will remain useful for very large simulations. Also, because the full matrix method requires the formation and inversion of an exact Jacobian, the time and memory required will scale, at least to some extent, with the square of the number of equations, while the

diagonal method scales nearly linearly with n_{eq}. This means that the diagonal method may be a more efficient tool for the solution of certain three-dimensional or reacting flow problems.

4. PARALLEL PERFORMANCE

The viscous DP-LUR method was written in FORTRAN90 and implemented on the 512 processor Thinking Machines CM-5 located at the University of Minnesota Army High Performance Computing Research Center. Each processor of this machine has four vector units, each with a peak performance of 32 Mflops; therefore the entire 512-node machine has a theoretical peak performance of 64 Gflops. Interprocessor communication is very slow compared to the computational speed of the processors, due primarily to the large latency. However, communication routines between nearest neighbors, such as CSHIFT, are highly optimized and are usually much faster than general router communication. Thus, it is necessary to minimize the total interprocessor communication in the algorithm and use the nearest neighbor routines whenever possible to obtain good performance.

Although the CM-5 supports both data-parallel and message-passing programming environments, the algorithm was implemented only in data-parallel, since as formulated it is inherently data-parallel and requires no asynchronous communication or computation. It is important to note that while it is relatively easy to modify a data-parallel code to run effectively on a message-passing machine by forcing the processors to act in lockstep with the use of synchronization directives, the reverse is not usually possible. Therefore this algorithm is quite general, and should run efficiently on a variety of parallel platforms with only minor modifications.

Both the diagonal and full matrix versions of the DP-LUR algorithm exhibit perfect scalability and high parallel efficiency.[3-4] The per-processor performance of each method increases with the number of points on each processing node, and is *independent* of the total number of processors. The peak per-processor performance of the perfect gas algorithms, obtained when all of the available memory is used, is about 40 Mflops for the full matrix method, and 34 Mflops for the diagonal version. The corresponding peak sustained performance on the full 512 node machine is then 21.3 Gflops for the full matrix method and 17.9 Gflops for the diagonal, which are both over 25% of the peak *theoretical* performance of the machine. The reacting flow algorithms are all about 30% slower, due to the evaluation of the exponentials required to form the reaction rates, which is a very inefficient operation on the CM5. The increase in performance for larger numbers of grid points per processor is due primarily to the vector nature of the machine; more points per processor implies a greater vector length.

5. CONCLUSIONS

The lower-upper symmetric Gauss-Seidel method of Yoon and Jameson has been modified to make it amenable to the solution of the Navier-Stokes equations on data-parallel machines. The resulting diagonal and full matrix data-parallel LU relaxation methods are both almost perfectly parallel, and display good stability and convergence properties on many perfect gas and reacting flows.

Both methods are implemented in data-parallel on the Thinking Machines CM-5, where they exhibit high parallel efficiency and perfect scalability. Their formulation makes the methods easy to implement in either data-parallel or message-passing modes, and therefore they should be portable to a variety of different parallel architectures. Floating point performance on the CM-5 is primarily a function of the number of grid points per processor, due to the vector nature of the machine. For very large perfect gas problems, the 512 processor CM-5 runs at about 17.9 Gflops for the diagonal method and 21.3 Gflops for the full matrix version, both of which are over 25% of the peak theoretical performance of the machine. The reacting flow codes are about 30% slower, due to an inefficient handling of the required exponentials. In addition, the memory usage of both methods is quite low, making it possible to run up to 32M grid points with the diagonal method and 24M grid points with the full matrix method for perfect gas problems on the 512 processor machine.

In short, the high parallel efficiency, low memory requirements, and numerical stability of the data-parallel LU relaxation methods make them potentially useful for the solution of very large perfect gas and reacting flow simulations.

REFERENCES

1. Simon, H. D., ed., *Parallel Computational Fluid Dynamics Implementations and Results,* MIT Press, Cambridge, MA, 1992.
2. Yoon, S. and Jameson, A., "An LU-SSOR Scheme for the Euler and Navier-Stokes Equations," AIAA Paper No. 87-0600, Jan. 1987.
3. Candler, G. V., Wright, M. J., and McDonald, J.D., "Data-Parallel Lower-Upper Relaxation Method for Reacting Flows," *AIAA Journal,* Vol. 32, No. 12, pp. 2380-2386, 1994.
4. Wright, M. J., Candler, G. V., and Prampolini, M., "A Data-Parallel LU Relaxation Method for the Navier-Stokes Equations," AIAA Paper No. 95-1750CP, June 1995.
5. Hassan, B., Candler, G. V., and Olynick, D.R., "The Effect of Thermo-Chemical Nonequilibrium on the Aerodynamics of Aerobraking Vehicles," *Journal of Spacecraft and Rockets,* Vol. 30, No. 6, pp. 647-655, 1993.
6. Tysinger, T. and Caughey, D., "Implicit Multigrid Algorithm for the Navier-Stokes Equations," AIAA Paper No. 91-0242, Jan. 1991.
7. Gnoffo, P. A., "An Upwind-Biased, Point Implicit Relaxation Algorithm for Viscous, Compressible Perfect-Gas Flows," *NASA TP 2953,* 1990.
8. MacCormack, R. W. and Candler, G. V., "The Solution of the Navier-Stokes Equations Using Gauss-Seidel Line Relaxation," *Computers & Fluids*, Vol. 17, No. 1, pp. 135-150, 1989.
9. Yee, H. C., "A Class of High-Resolution Explicit and Implicit Shock Capturing Methods," *NASA TM 101088,* 1989.
10. Park, C., Howe, J. T., Jaffe, R. L., and Candler, G. V. "Review of Chemical Kinetic Problems of Future NASA Missions, II: Mars Entries," *Journal of Thermophysics and Heat Transfer,* Vol. 8, No. 1, pp. 9-23, 1994.
11. Yoon, S. and Kwak, D., "Multigrid Convergence of an Implicit Symmetric Relaxation Scheme," *AIAA Journal,* Vol. 32, No. 5, pp. 950-955, 1994.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel Simulation on Rayleigh-Bénard Convection in 2D by the Direct Simulation Monte Carlo Method

Mitsuo Yokokawa[a], Dave Schneider[b], Tadashi Watanabe[a], and Hideo Kaburaki[a]

[a]Center for Promotion of Computational Science and Engineering,
Japan Atomic Energy Research Institute,
2-28-8, Hon-Komagome, Bunkyo-ku, Tokyo 113, Japan.

[b]Cornell Theory Center, Engineering and Theory Center Bld.
Ithaca, NY 14853-3801, U.S.A.

The Direct Simulation Monte Carlo method is one of the powerful methods to obtian macroscopic properties of flows from the microscopic motion of particles. The parallel DSMC code has been developed to simulate two dimensional Rayleigh-Bénard convection by using a message passing system PVM3. The speedup of 7.9 going from 2 to 16 processors is obtained by the parallel code with the adaptive scheme for defining collision cells on the IBM SP2.

1. Introduction

The direct simulation Monte Carlo (DSMC) method is frequently used in gas kinetics for simulating a wide range of flows from the rarefied to the near continuum region. This technique was developed by Bird [1] and has been applied to various kinds of flows. This method obtains macroscopic properties such as temperature and velocities by averaging microscopic motions of simulated particles over small volumes. Therefore, the method is suitable for deriving microscipic characteristcis from macroscopic flows.

Several studies for the Rayleigh-Bénard convection have been carried out using the DSMC and MD methods and conducted the convection rolls at high Rayleigh number. In almost all of those studies, however, semislip boundary conditions are applied at the top and bottom walls. Stefanov and Cercignani [2], and Watanabe, et al. [3] have been successful in simulating rolls with the diffuse reflection boundary condition. In particular, Watanabe, et al. have derived the critical Rayleigh number for the Rayleigh-Bénard system in the continuum region using the two-dimensional DSMC simulation with diffusive boundary conditions in a rectangular region with aspect ratio of 2.016 and discussed the influence of the boundary condition on the field variables.

For the simulation of flows in the continuum region described by the conventional hydrodynamics, a large number of simulated particles are needed to reduce statistical errors and reproduce physically meaningful flow patterns. When the Knudsen number is small, the mean free path is small compared with the characteristic length scales and a large number of collision cells are required. Moreover, since the time step has to be equal

to or less than the mean free time between collisions, a large number of time steps are necessary to reach a steady state flow. These considerations indicate that the simulations of two-dimensional flow with larger aspect ratios and three-dimensional continuum flows by the DSMC method require more computational power and memory capacity than are currently available on serial and vector mainframe computing platforms.

One of the solutions to reduce the computational time is parallel computation. In the DSMC method, collisions between simulated particles are restricted only within a small region called a collision cell during each time step. It is clear that the collision processes in each cell are independent, thus can be carried out in parallel. Yokokawa, et al.[4] have developed a parallel code of the DSMC simulation on the Fujitsu AP1000 and have evaluated the efficiency of parallel processing for this class of problems. In these studies, the computational domain is divided into subdomains which have the same number of collision cells and computation of the particles in a subdomain is assigned to a processor.

In this study, we have developed several portable parallel DSMC codes for simulating flows in a rectangular region using the PVM (Parallel Virtural Machine) message passing library. The codes were developed on the IBM SP1 and SP2 systems at the Cornell Theory Center.

2. The DSMC Method

The DSMC method consists of two distinct processing steps. The movement process describes the evolution of the system between collisions, and the collision process describes the effects of collisions between simulated particles. The computational domain is divided into collision cells in which the interchange of momenta for simulated particles takes place. The length of the collision cell is taken to be nearly equal to the size of the mean free path. The Bird's time counter method is used for the collision process and the hard sphere model for the particles and the elastic collision model for the momentum exchange are assumed in the implementation.

The sampling cells, in which physical quantities such as density and velocities are derived, are defined as groups of collision cells and can be taken independently of the collision cell. The flow properties of the computational region are calculated by taking the time averages of the number of simulated particles and their velocities through several time steps.

Since the size of the collision cell must be the same order as the mean free path, it is sometimes difficult to discretize the region by fixed collision cells with appropriate size. An adaptive scheme for defining collision cells [5] has been implemented in the code. In this scheme, each sampling cell is divided into several collision cells automatically with appropriate size to the local mean free path in the sampling cell as simulation proceeds.

3. Parallelization

As seen in the algorithm of the DSMC method, collisions between simulated particles are restricted only within a collision cell during each time step. It is clear that the collision processes in each cell are independent, thus can be carried out in parallel. In the previous parallel implementation on the AP1000[4], the computational domain is divided into subdomains so that each processor is assigned nearly equal numbers of collision

cells. The assignment of the subdomains to processors is determined in the beginning of simulation and a reference table between sampling cells and processors (hereafter called as the SC-PE table) is created on all processors. The sampling cell in which a simulated particle resides is determined by a simple binning operation using its spatial coordinates, and the processor in which this sampling cell resides is determined by referring to the SC-PE table. If the particle moves from one subdomain to another, the data for this particle is transferred between processors. The dynamic allocation can be applied to keep the load balance by the mean that the SC-PE table is reconfigured dynamically in all processors while the simulation is being carried out.

The transfer of particles between processors occurs because particles move throughout the whole computational domain. The rate of particle transfers between processors can be quite high in the convective flow regime. Initially, two barrier synchronizations were used in the particle data transfer process. The first synchronization was just after sending the number of particles to the processor which is expectd to receive the particle data, and the second one after sending the particle information (positions and velocities). Subsequently, we developed a loosely synchronous implementation which does not require any global barrier operations.

The statistics taken in each processor are collected on a processor at some interval through the simulation and written out onto the file by the processor. The whole particle information is also saved into the file at the end of a run so that the sequence of jobs can be executed.

The PVM3 system is used for the parallelization and the SPMD programming model is taken to make parallel DSMC codes on the IBM Powerparallel system SP1 and SP2.

4. Computational Results and Performance

Two simulations on the lid-driven cavity flow and the Rayleigh-Bénard convection have been carried out to check the validity of the parallel codes.

The computational domain is divided into 64×64 sampling cells for the lid-driven cavity flow. The initial temperature and pressure are 283K and 0.002Pa, respectively, which correspond to the air at an altitude of 100km. The Knudsen number is 0.005 and the Reynolds number is 100. The simulated particles of 320 are assigned initially in a sampling cell. The particles of 1,310,720 in total are used in the simulation. The adaptive scheme is not applied in the simulation. The simulation result is in good agreement with that of Navier-Stokes simulation. The performance of 7.14, 3.45, 1.78, and 1.06 μsec/particle/iteration has been obtained with 2, 4, 8, and 16 processors, respectively, on the IBM SP1. This corresponds to a speedup of 6.7 going from 2 to 16 processors.

The results of numerical simulation on Rayleigh-Bénard convection of air by the parallel code have been compared with those by Watanabe and others [3]. The simulation domain is a two-dimensional rectangle, the aspect ratio of which is 2.016. The height is 5.6mm. The initial temperature and pressure are 80K and 20Pa, respectively. The Knudsen number is about 0.016. This domain is divided into 40×20 sampling cells and 400 particles are initially placed in each sampling cell. The Rayleigh number is varied from 2000 to 5000, and the bottom wall temperature corresponding to the Rayleigh number is placed in the beginning of the simulation. The samplings is taken every 2 time step. The

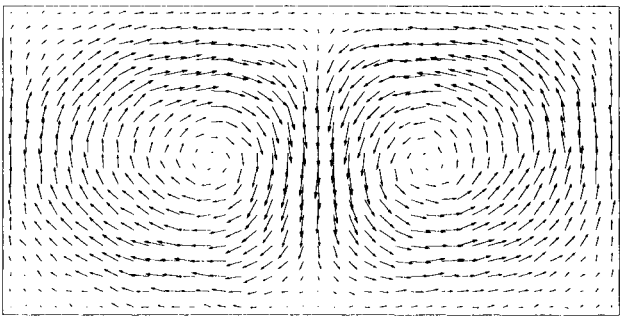


Figure 1. Velocity profile in the steady state at Ra=3000.

adaptive scheme of generating collision cells is applied for this problem. Each sampling cell is divided into 25-36 collision cells according to the local mean free path.

Figure 1 shows the convection rolls at Rayleigh number of 3000 in the steady state. The rolls are generated after 4000 time steps and kept in a stable state. Though the direction of the vortex is downward flow in the center of the cavity in this case, it is possible to obtain upward flow in the center due to the difference of random number sequences.

The comparison of the result by parallel calculation with that by serial one is made and the time development of the mid-elevation temperatures of both calculations at Rayleigh number of 3000 is shown in Figure 2. The result by the serial code is denoted by the lines tagged 'Serial' and so is the result by the parallel code by the lines tagged 'Parallel.' The lines with 'Avg', 'Left', and 'Cntr' in the figure mean that the average temperature along with the horizontal line, the temperature sampled at the left-most sampling cell and the center cell, respectively. The temperatures are normalized so that the tempertures of the top and bottom walls are 0.0 and 1.0, respectively. The exact agreement was not obtained between the codes. It seems that the random number sequences used in the simulation are different between in serial calculation and in parallel one. However, the tendency of the temperature development is almost the same in both calculations. The relation between temperatures and the Rayleigh number are also in good agreement with those obtained by Watanabe, et al. Figures 3 and 4 show the CPU time measured in the main process on the SP1 and SP2 as the number of processors increases as 2, 4, 8, and 16, respectively. The high performance switch is taken as a communication network between processors on both system. Three different sizes of data are considered in the measurement, which are 200, 400, and 800 particles in a sampling cell and 160000, 320000, and 640000 paticles in total in a whole computational region. The 20 time steps have been advanced with 10 samplings. The symbols '$\diamond$' and '+' mean the total CPU time and the time within the iteration loop of time advancement, respectively. It is found that

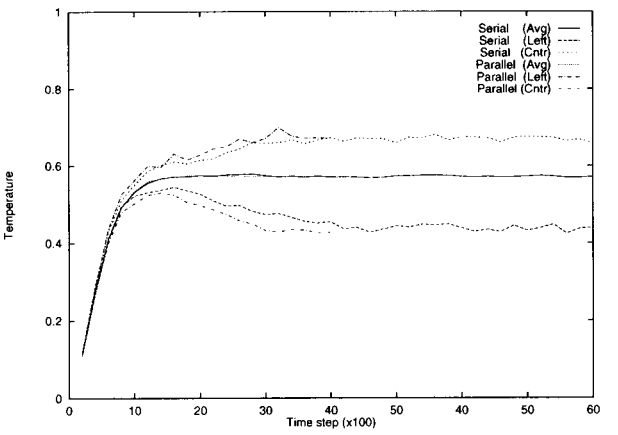


Figure 2. Comparison of time development of the mid-elevation temperature at the Rayleigh number of 3000 between the serial and the parallel codes.

the CPU time decreases at the rate inversely proportional to the number of processors. The performance of 11.40, 5.84, 2.83, and 1.45 μsec/particle/timestep has been obtained with 2, 4, 8, and 16 processors, respectively, on the IBM SP2. This corresponds to a speedup of 7.9 going from 2 to 16 processors. It is clear that the parallel code has the scalability. However, the performance for the Rayleigh-Bénard simulation is slightly lower than that for the cavity flow simulation. It seems that the adaptive scheme fro generating the collision cells needs more CPU time to define the collision cell dynamically during the simulation. The CPU time on the SP2 is 30% faster than that on the SP1.

The CPU time required to save the information of all partilces for restart jobs, which are the locations and velocities of all particles, is also measured. The amount of data are 6.4, 12.8, and 25.6 megabytes for the number of particles of 160000, 320000, and 640000, respectively. The data is archived into a file in the main process and the data on the other processors should be transferred to the main process before saving. The CPU time increases as the number of processors increases because the time required for the data transportation increases. The performance of nearly 3MB/sec and 2MB/sec for saving data onto a file is measured the SP1 and SP2, respectively.

5. Conclusion

We have developed the serial and parallel DSMC codes with the adaptive scheme for defining the collision cells. The parallel code is implemented with the SPMD programming model by using the PVM3 message passing library. It is found that the results obtained by these codes are in good agreement with the previous ones. The performance of 11.40, 5.84,

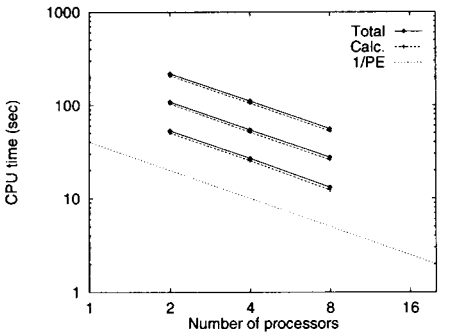


Figure 3. CPU time on the SP1

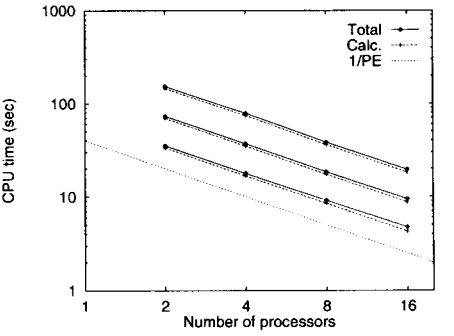


Figure 4. CPU time on the SP2

2.83, and 1.45 μsec/particle/timestep has been obtained with 2, 4, 8, and 16 processors, respectively, on the IBM SP2 in the case that the adaptive scheme for defining collision cells is used. This corresponds to a speedup of 7.9 going from 2 to 16 processors. It is clear that the parallel code has the scalability.

Acknowledgments

This research was peformed at the Cornell Theory Center while the first author stayed there as a visiting scientist. The authors would like to thank Profs. Malvin Kalos and Marcy Rosenkrantz for allowing us to use the facilities. They also would like to thank Drs. Steve Hotovy and Jerry Gerner for their helpful discussion.

REFERENCES

1. G. A. Bird, Molecular Gas Dynamics, Oxford University Press, Clarendon, Oxford (1976).
2. S. Stefanov and C. Cercignani, European Journal of Mechanics B Fluids, Vol.11, 543 (1992).
3. T. Watanabe, H. Kaburaki, and M. Yokokawa, "Simulation of a two-dimensional Rayleigh-Bé nard system using the direct simulation Monte Carlo method," Physical Review E, Vol.49, No. 5, pp.4060-4064 E (1994).
4. M. Yokokawa, K. Watanabe, H. Yamamoto, M. Fujisaki, and H. Kaburaki, "Parallel Processing for the Direct Simulation Monte Carlo Method," Computational Fluid Dynamics JOURNAL, Vol.1, No.3, pp.337-346 (1992).
5. H. Kaburaki and M. Yokokawa, "Computer Simulation of Two-Dimensional Continuum Flows by the Direct Simulation Monte Carlo Method,", Molecular Simulation, Vol.12(3-6), pp. 441-444 (1994).

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

Distributed Implementation of KIVA-3 on the Intel Paragon

O. Yaşar[a*] and T. Zacharia[b*]

[a]Center for Computational Sciences, Oak Ridge National Laboratory,
P.O. Box 2008, MS-6203, Oak Ridge, TN 37831

[b]Metals and Ceramics Division, Oak Ridge National Laboratory,
P.O. Box 2008, MS-6140, Oak Ridge, TN 37831

We describe a message-passing implementation of KIVA-3 combustion engine code. Although the pre-processor uses a block-structured mesh to begin with, the main code does not require any order in the grid structure. Each computational element is recognized through the neighborhood connectivity arrays and this leads to indirect addressing that complicates any type of loop decompostion on parallel machines. Dependencies in the code extend only one layer in each direction and the presence of ghost cells and cell-face boundary arrays in the code suits block-wise domain decompostion very well. Internal boundaries are handled like the external boundaries, enabling one to apply the same rules, physics and compuatation to a part of of the domain as well as the whole domain. The code is currently being tested on the Intel Paragon at CCS-ORNL. The parallel efficiency for a simple baseline engine problem has shown to be more than 95 % on two nodes, a much more optimistic performance indicator compared to earlier KIVA-2 implementations. Further work is underway to test the validity and scalibility of the algorithm for a range of problems.

1. Introduction

KIVA-3 [1] comes as a package comprising of a pre- and post-processor and the hydro code. The pre-processor uses a block-structured mesh to represent the physical domain and that provides an ability to model complex geometries. The physical domain is surrounded by ghost cells in all directions, a significant difference from KIVA-2 [2,3]. The introduction of ghost cells in such abundance increases the number of mesh points in one hand, but it gives the ability of dividing the global domain into small independent blocks, easing the constraints of having a large tensor-product grid to represent complex geometries. Each block is initially created independently using a tensor-product grid, but they are all patched together at the end with connectivity arrays describing what the surrounding points are for each mesh point. Thus, the outcome from the pre-processor is somehow similar to unstructured meshes and the hydro code will accept mesh created

*The work was sponsored by the Division of Material Sciences of the U.S. Department of Energy, and the Center for Computational Sciences (CCS) of the Oak Ridge National Laboratory under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc.

with any pre-processor as long as the connectivity arrays are provided. The fact that the grid points do not have to be stored in any order in data arrays leads to an ability to sort out the active and non-active (ghost) cells and that eventually leads to shorter vector lengths in the computational loops.

The use of connectivity arrays and indirect addressing is advantageous, but there are disadvantages associated with indirect addressing as well, particularly for parallel implementations that are based on loop decomposition. The only way to counter that is to have each processor localize its computation to a group of mesh points with the least number of references to remote points. In other words, instead of dividing the loop among processors, we divide the problem geometrically such that each processor's computation is localized to a block of meshes that are in physical contact. Indirect addressing is not a problem for processors to handle elements in their own region so long as they can identify the elements that must be exchanged with the neighbor processor during communication.

Altogether; the use of ghost cells, connectivity arrays and cell-face boundary conditions in all directions creates a general recipe of physics, numerics and boundary conditions that can be applied to a part of the domain as well as the whole domain, and thereby providing for a convenient block-wise domain decomposition. The following sections will further explain the implementation of such a block-wise distribution for data and computation on distributed-memory parallel computers. Following a section on the grid and data structure, we will discuss the dependencies for diffusion, advection and spray dynamics taking place in the computational volumes. Finally, a simple test problem and its parallel execution on the Intel Paragon will be described.

2. Grid and Data Structure

KIVA-3 is a 3D finite-difference code, solving fluid, particle and chemistry equations over a mesh made up of arbitrary hexahedrons. Each hexahedron is a cell with vertices located at the corners. A regular cell is represented by its lower left front vertex. A typical representation of the block-structured grid system for KIVA-3 is shown in Figure 1. The data (grid locations, physical quantities) for such a tensor-product grid could be stored in 1D arrays by a single index that increases in an x-y-z order sweeping x-y planes one at a time. Data stored in this fashion will still maintain the structure of the grid, and neighborhood connectivity is defined through the x-y-z ordering of the cell index. A computational loop over the entire mesh will sweep the elements of 1D arrays from the start to the end including non-active cells represented by the vertices on the boundaries. Since boundary vertices do not represent real cells, this might create a significant loss in execution time when there are many of such non-active cells. KIVA-3 addresses this problem by storing the connectivity information into separate arrays for each direction and using indirect addressing through these arrays to identify the neighbor points in the grid. Since the connectivity information is kept independent of the initial x-y-x ordering of the data storage, the ordering can be compromised in favor of grouping active and non-active cells separately using the flags described below. The storage of array elements according to such sorting in KIVA-3 is illustrated in Figure 2.

The real physical domain is surrounded by ghost cells in all directions. The boundary vertices on the right, back, and top are real vertices, but they represent ghost cells that are

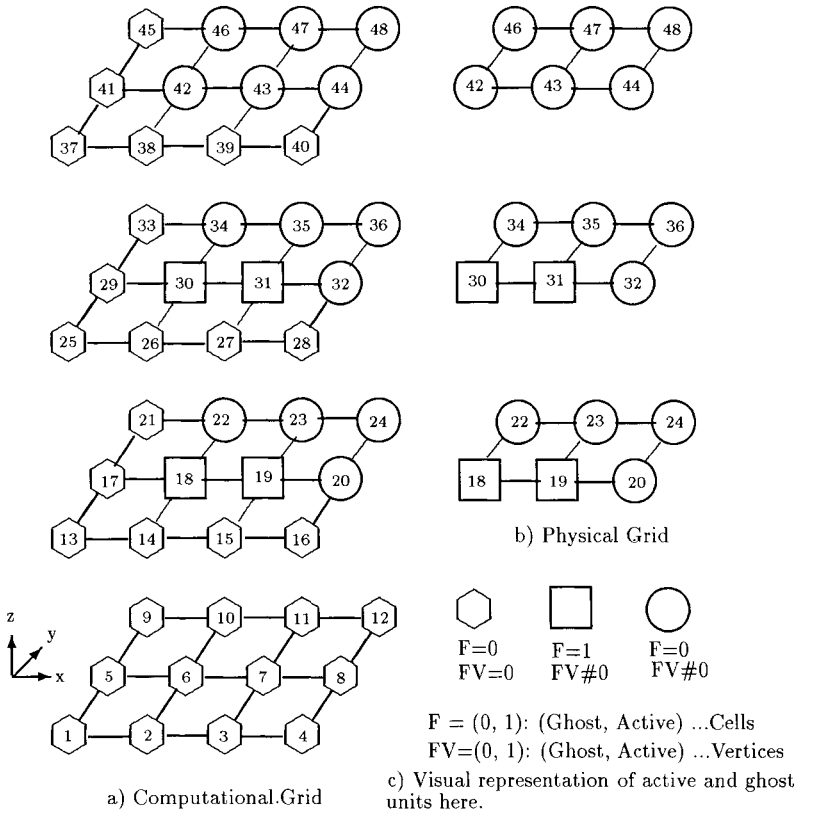


Figure 1. Grid Structure as shown in x-y planes.

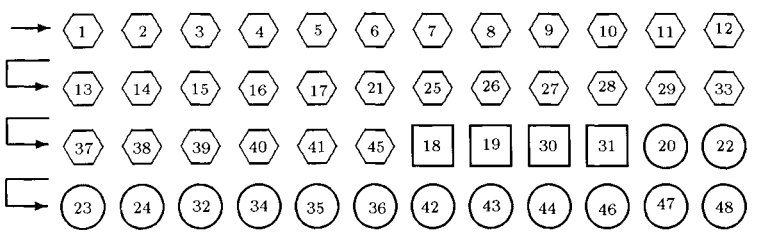


Figure 2. Storage sorted by F and FV to maintain the shortest possible vector lengths.

not part of the physical domain. There are flags (F for cells, FV for vertices) associated with each cell and vertex to indicate if they are real or not. The flags take values of 1 or 0. Based on the flag values one can sort out the elements on such a grid in different groups. Elements with $F = 0, FV = 0$ correspond to the ghost cells on the left, front, and bottom, whereas elements with $F = 1, FV = 1$ correspond to the active cells, elements with $F = 0, FV > 0$ correspond to ghost cells on the right, back, and the top as distinguished through different shapes in Figure 1 and 2.

In a distributed implementation, the ghost cells could match the real cells of outer layers residing on other adjacent processors. Each processor would then copy information from adjacent processors into these ghost cells and treat this information just like the boundary conditions. Since the access pattern for vertices and cells on the boundaries need to be known in advance for communication requirements, one needs to further sort out active and non-active cells within themselves separating the ones on the left, right, and so on. One would also have to assume that the boundary shared between neighbor processors has the same grid points on both sides to assure proper communication. The suggested sorting could be either done in the pre-processor or the main program itself.

Although a 3-D domain decomposition is considered as the ultimate goal here, the current thinking is towards a one-dimensional one. The piston motion in z-direction suggests that decomposition of blocks be parallel to the z-direction for best possible load balancing and low communication/computation ratios. During the piston motion, the grid should be decomposed, re-organized and sorted again as done in the beginning of the simulation. There is need for processor communication to assure the removal of the grid at the same $x - y$ planes. The interface between processors is assumed to have the same grid points on both sides to match the communication patterns when needed. A more general approach would eliminate this requirement by gathering into the ghost cells values computed by interpolations or extrapolations.

3. Time and Spatial Dependency

The governing equations for the fluid and particle dynamics and chemical reactions are discretized in both time and space in KIVA-3. The temporal differencing is done in 3 steps (A,B, and C). Phase A is where most of the source terms are calculated, Phase B is where the diffusion terms are calculated and Phase C is where the advection is computed. KIVA-3 uses a

$$\rho^{n+1} = \phi_D \cdot \rho^{n+1} + (1 - \phi_D) \cdot \rho^n \tag{1}$$

type formula mixing both explicit and implicit schemes to adapt the solution to the changing CFL condition. Here, ϕ_D is the implicitness variable and is computed at every time step to check the flow conditions. Implicit schemes generally pose difficulties for parallel computing due to the dependencies on the current values of the variable that is being calculated. However, the implicit solvers in KIVA-3 are iterative ones and the physical variables on the RHS of the governing equations are already known (predictor) and thus require no sequential dependencies between processors. However, the processors have to march together and make available to each other the necessary values at each iteration.

Discretized equations for mass density, specific internal energy and turbulence equations are solved for cell-centered quantities and thus require both cell-center and cell-face values of variables involved. Most of the physical quantities are cell-averaged and are computed via control volume approach. The momentum field is computed via momentum cells that are centered around vertex locations. Cell-face velocities are computed at cell faces via cell-face volumes centered around the face it involves. Volume integrals of gradient terms are converted into surface area integrals using the divergence theorem and computation of such terms constitute the heart of the diffusion solvers in the code. Such area integral terms over surfaces become sums over cell faces:

$$\int_S \nabla Q \cdot d\mathbf{A} = \sum_{\alpha} (\nabla Q)_\alpha \cdot \mathbf{A}_\alpha \tag{2}$$

where α represents one of the six cell faces. Evaluating the gradient on the surface is done as follows

$$(\nabla Q)_\alpha \cdot \mathbf{A}_\alpha = \alpha_{lr}(Q_l - Q_r) + \alpha_{tb}(Q_t - Q_b) + \alpha_{fd}(Q_f - Q_d), \tag{3}$$

where t, b, f, d indicate top, bottom, front and back mid-points on the cell face α, and l, r indicate the cell centers on the left and right sides of the cell face. Here, α_{lr}, α_{tb}, and α_{fd} are geometric factors for face α in $left - right, top - bottom, front - back$ directions, and $Q_{t,b,f,d}$ values are averages computed at mid-points on this face. Averaging is done over the values of Q in the four cells surrounding cell edge in question. Q_l and Q_r are values at cell centers on the left and right sides of the cell face α. Since the evaluation of variables on the cell face mid-points is done through averaging among the cells that share the cell face in question, cell-face averaging is a potential for communication between adjacent processors for the cell faces on the boundary.

The momentum equation is solved at vertex locations and thus requires quantities at both vertex locations and momentum-cell faces. Volume integrals over momentum cells are also converted to area integrals. Momentum cells are constructed around vertices and involve 8 regular cells that share a particular vertex. There are a total of 24 faces (β), three of each reside in one of the 8 cells. The vertices on the boundary do not have as many β faces as the internal ones unless the boundary is shared by other processors that have the remainder of the missing faces. Evaluating quantities on the boundary momentum cell faces will involve processor communication. Contributions from other processors also need to be gathered for physically internal but computationally external boundary vertex momentum cells.

Cell-face velocities require the pressure evaluation on both sides of the regular cell faces. A cell-face volume is centered around the face in question with faces (γ) cutting through the cell centers on both sides. Again the computation for cell-face velocities has to collect contributions from adjacent processors that share the face in question.

A block-wise decomposition (whether in one or multi-direction) requires the processors share a common face. This in no way indicates that there is a duplication by the adjacent processors of the vertices on the common faces. A vertex gets 1/8th of its mass from each cell that has the vertex in common, and vertices on the processor boundary are only partially represented by each processor. KIVA-3 applies the same rules (physics, numerics and BCs) to every computational cell, except some of the external boundary

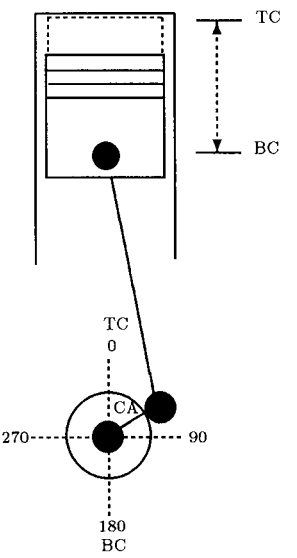


Figure 3. Basic geometry of the reciprocating internal combustion engine.

conditions are applied via the F flags. Care needs to be taken to treat physically internal but computationally external boundaries as internal by temporarily changing the flags to 1.0.

Advection of cell-center quantities require computing cell-face quantities and that in turn requires cell-center quantities and their derivatives on both sides of the face depending on the upwind conditions. Required information is gathered from other processors and put into ghost cells. Advection of momentum, however, is a somewhat different. One not only has to gather derivatives and variable values from vertices in the neighborhood, but also has to sum the contributions to account for all the momentum cell faces residing in multiple number of regular cells. One can use either Quasi Second Order Upwind (QSOU) or first order Partial Donor Cell (PDC) Upwind to evaluate the quantities on the cell faces. For cell-centered variables (mass, energy, etc) this requires derivatives at cell centers on either sides of the face. For momentum, this requires derivatives of velocity field at vertex locations on both sides of the vertex at which the momentum cell is located.

The governing equations for spray dynamics and chemical reactions are discretized over the same mesh system as we have described so far. The fluid flow is affected by the spray injection and chemical ignition processes and the effects show up in the fluid equations, mostly in the source-term. Spray injection and fuel ignition processes are both local and may not occur in the domains of all the processors, causing a slight load-balancing problem. Most of the spray and chemistry quantities are cell-centered and involve no dependencies between neighboring cells. Spray particles cross cell faces and may end up

changing processor domains. A mechanism for particle destruction and creation across processor boundaries is required. Spray particles contribute to the vertex masses and momentum and their interaction with fluid is described through collisions, breakup, and evaporation.

4. Test Problem

A typical problem to test our parallel algorithm would be a reciprocating engine, where the piston moves back and forth in a cylinder and transmits power through a connecting rod and crank mechanism to the drive shaft as shown in Figure 4. Our current capability does not include a dynamic grid yet, thus the piston motion cannot be tested yet at this stage. Also, the spray injection is turned off to separate the load balancing effects from the initial focus on the fluid flow and combustion. The piston chamber is modeled with a 20x1x20 (x-y-z) cylindrical grid using the pre-processor separately on two identical blocks for a two-node execution. Each node then works on a 10x1x10 physical grid. The computational grid includes one extra-layer (buffer zone) in each direction to accommodate the boundary conditions and interprocessor communication. The grid data (*tape*17) needed by the main code can be generated through any pre-processor as long as the grid points, their connectivity arrays follow required format. Generating this data for different domain partitioning and varying number of nodes will require a little bit of effort compared to cases where decomposition and the number of nodes are determined during run time.

The run time was set up to 100 cycles with t=14 seconds. The processors seemed to march together with the same Δt time step as if the off-site partition were running along with the resident partition. The results in *tape*12 seemed consistent with each other and also with the single-node output. A comparison of graphical output *tape*9 shows a consistent continuity in pressure contours between the two partitions that is also similar qualitatively with the single-node run. The parallel speedup for both first order (PDC) and second-order (QSOU) upwind differencing is shown in Figure 4. Though PDC is normally a cheaper method, it is less amenable to parallelism due to the Robin-Hood type correction of negative fluxes introduced through first order differencing. If a cell ends up with a negative mass, then the neighbor cells with positive values have to bring that to at least zero by giving away from their own. If such a cell happens to be on the boundary then this will involve interprocessor communication.

5. Summary

A block-wise decomposition approach is described for implementing KIVA-3 on distributed memory machines. Dependencies in the code extend only one layer in each direction and the presence of ghost cells and cell-face boundary arrays in the code suits a distributed-memory implementation well. There seems to be no dependency created through temporal differencing since variables are computed based on the quantities from the previous iteration or time step. Spatial differencing requires estimating variables and sometimes their gradients (diffusion terms) on the cell faces which leads to communication between adjacent processors sharing the cell-face in question. Momentum cells around the boundary vertices are split between processors, requiring each to compute only their share of the vertex momentum contribution. Advection involves fluxing through regular and momentum

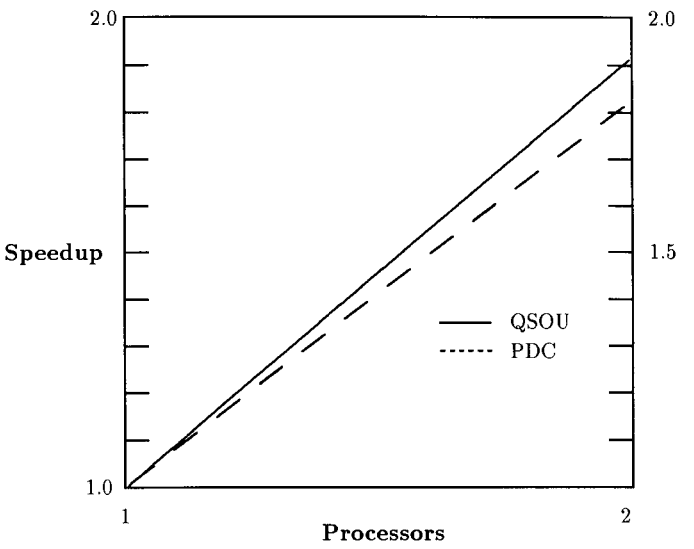


Figure 4. Speedup results for the test problem.

cell faces. Cell-face values that are evaluated via upwind differencing (QSOU or PDC) require physical quantities and their derivatives on both sides of the face. Spray dynamics and chemistry are expected to require little communication but non-uniform distribution of particles will lead to slight load balancing problems. Particles cross processor boundaries and need to be created and destroyed. The grid points on the shared faces between processors need to have the same structure for predictable communication patterns. The testing of the algorithms is being done on the Intel Paragon and the speedup of 1.9 have been obtained for a simple baseline engine case on 2 processors. Further results including spray dynamics, and piston motion for more general problems are expected in the near future.

REFERENCES

1. Amsden, A. A. 1993. "KIVA-3: A KIVA Program with Block-Structured Mesh for Complex Geometries." Technical Report LA-12503-MS. Los Alamos National Laboratory, Los Alamos, NM 87545.
2. Amsden, A. A., P. J. O'Rourke, and T. D. Butler 1989. "KIVA-II: A Computer Program for Chemically Reactive Flows with Sprays." Technical Report LA-11560-MS. Los Alamos National Laboratory, Los Alamos, NM 87545.
3. Yaşar, O. and Christopher J. Rutland 1992. "Parallelization of KIVA-II on the iPSC/860 Supercomputer." *Proceedings of Parallel Computational Fluid Dynamics '92 Conference* (New Brunswick, NJ, May 18-20), 419-425.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A PARALLEL TOOL FOR THE STUDY OF 3D TURBULENT COMBUSTION PHENOMENA

A. Stoessel * [a]

[a]Institut Français du Pétrole, F-92506 Rueil-Malmaison, France, Alain.Stoessel@ifp.fr

Direct Numerical Simulation of 3D reactive turbulent flow is a very useful tool for a better knowledge of turbulent phenomena, but accurate and costly numerical methods have to be used in order to capture all physical scales. A modern efficient tool named NTMIX3D has thus been built on MPP platforms and released to physicist users to perform some real practical studies.

We will present in this paper our experience in building this new tool on a wide range of computers including modern distributed memory parallel computers. After a brief statement on the physical problem and its numerical implementation, we will describe the parallel computer implementation including dynamic load balancing and insist on numerical improvements and software engineering used to achieve high efficiency. Lastly, we will show a practical simulation of typical turbulent combustion phenomena.

1. Physical Problem and Parallel Implementation

Predicting and controlling pollution by flames (piston engines, furnaces, heaters, aircraft engines) will be some of the great challenges of combustion research and industrial development in the next years. This prediction has to be done in most cases for turbulent flames for which very little fundamental experience exists on the mechanisms controlling pollutant formation. This situation is quite different from the one encountered in laminar flows for which many research groups have developed efficient tools to predict flame structure and pollutants for any chemical scheme. Unfortunately the information obtained in laminar situations can not be applied directly to most practical combustion devices because these systems are controlled mainly by turbulent motions. New methods devoted to the problem of chemistry - turbulence interactions have to be developed.

Recently, Direct Numerical Simulation of turbulent flames has emerged as one of the major tools to obtain information on turbulent flames and to be able to construct models for those turbulent reacting flows [1–3]. In these methods, the reacting flow is computed using a high-precision numerical scheme, an accurate description of chemistry and no model for turbulence.

The present DNS method is solving the full 3D compressible Navier-Stokes equations

*This work is partially supported by the Centre de Recherche en Combustion Turbulente (IFP - ECP/CNRS - CERFACS).

with variable transport coefficients and simple or complex chemistry:

$$\frac{\partial \rho^*}{\partial t^*} + \frac{\partial}{\partial x_i^*}\left(\rho^* u_i^*\right) = 0$$

$$\frac{\partial m_i^*}{\partial t^*} + \frac{\partial}{\partial x_j^*}\left(m_i^* u_j^*\right) + \frac{\partial p^*}{\partial x_i^*} = \frac{1}{Re} \cdot \frac{\partial \tau_{ij}^*}{\partial x_j^*}$$

$$\frac{\partial e_t^*}{\partial t^*} + \frac{\partial}{\partial x_i^*}\left[\left(e_t^* + p^*\right) u_i^*\right] = \frac{1}{Re} \cdot \frac{\partial}{\partial x_j^*}\left(u_i^* \cdot \tau_{ij}^*\right) - \frac{1}{Re \cdot Pr} \cdot c_p^* \frac{\partial q_i^*}{\partial x_i^*} + \alpha' T_\infty^* \rho^* \tilde{Y} RePr \cdot \dot{\omega}$$

$$\frac{\partial\left(\rho^* \tilde{Y}\right)}{\partial t^*} + \frac{\partial}{\partial x_i^*}\left(\rho^* \tilde{Y} u_i^*\right) = \frac{1}{ReSc} \cdot \frac{\partial}{\partial x_i^*}\left(\mu^* \frac{\partial \tilde{Y}}{\partial x_i^*}\right) - \rho^* \tilde{Y} RePr \cdot \dot{\omega}$$

where

$$e_t^* = \frac{1}{2}\rho^* \cdot \sum_{k=1}^{3} u_k^{*2} + \frac{p^*}{\gamma - 1} \qquad p^* = \rho^* T^* c_p^* \frac{\gamma - 1}{\gamma}$$

$$\mu^* = \mu_0^* \cdot \left[\frac{T^*}{\frac{T_0}{T_\infty}} \cdot (\gamma - 1)\right]^b \qquad \tau_{ij}^* = \mu^* \cdot \left(\frac{\partial u_i^*}{\partial x_j^*} + \frac{\partial u_j^*}{\partial x_i^*} - \frac{2}{3}\delta_{ij}\frac{\partial u_k^*}{\partial x_k^*}\right)$$

Equations are written in a dimensionless form with Einstein convention for the summations. ρ^* is the gaz density, u_k^* are the velocity components, m_k^* the momentum components, e_t^* the total energy and $\tilde{Y}$ the mass fraction of fresh gaz. p^*, T^* and μ^* are respectively the pressure, the temperature and the viscosity. q_i^* is the heat flux, τ_{ij}^* is the Reynolds stress tensor, $\dot{\omega}$ is the production term coming from the one step chemical reaction and α' is the temperature factor of this reaction. Re, Pr and Sc are respectively the Reynolds number, the Prandtl number and the Schmidt number.

The system is discretized by a finite difference method on a regular or irregular cartesian grid. The code makes use of a 6^{th} order compact scheme (Padé scheme) for spatial derivation [4] and a 3^{rd} order Runge-Kutta method for time integration. The low-dispersive and non-dissipative properties of these schemes allow to handle most of the different scales that are existing in turbulent combustion phenomena. NSCBC method [5] provides accurate boundary condition treatments for this kind of codes even with non-reflecting boundaries such as physical open boundaries.

2. Computer Aspects and Software Engineering

The use of 6^{th} order compact schemes for the computation of space derivatives implies the frequent resolution of tridiagonal linear systems (135 times per time step for 3D) with multiple right hand sides (RHS) . These systems are alternatively laying on the 3 different mesh directions X, Y and Z. The less expensive method for solving such systems is LU factorization with precomputed coefficients, better known in CFD as Thomas' algorithm.

Even if on shared memory computers parallelization is automatic by spreading the different RHS solution on the different processors, the need of data locality makes the work much more complex on distributed memory systems. Various methods have been tested on different platforms and with different programming models [6] but domain decomposition methods as explained in [7] have been chosen because achieving high efficiency on a wide range of MPP platforms. These methods are requiring some overlapping between the

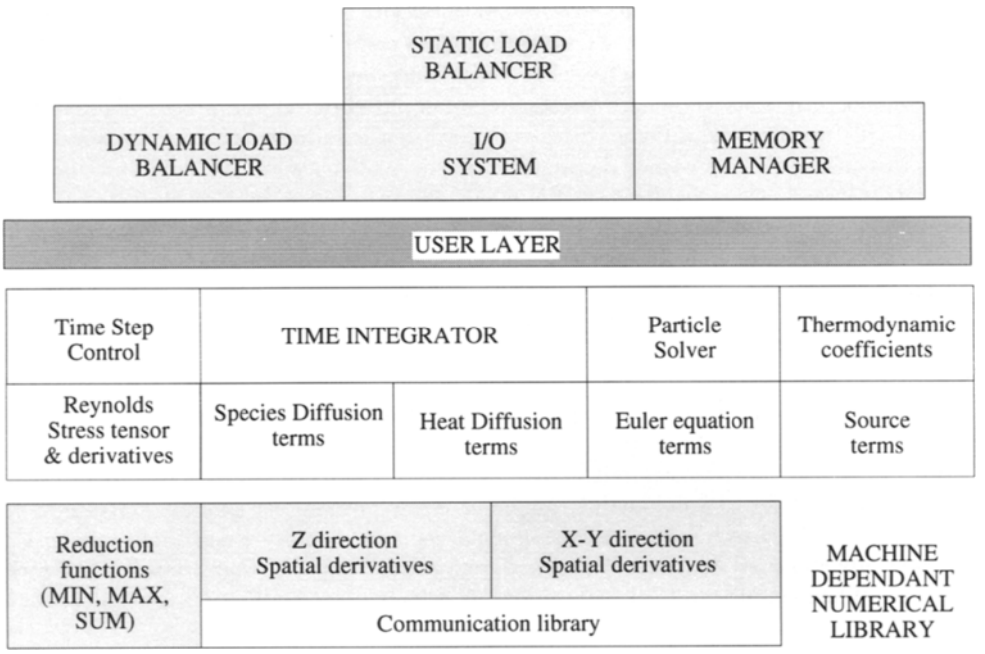


Figure 1. NTMIX3D Building blocks

different subdomains, a computation overhead is thus existing because of redondant computations. Depending of the communication performance of each MPP, we can choose between a method which is requiring more communications and less overlapping or a method with very few communication but more overhead. In the first case, communications are fully overlapped by computations if the hardware and the message passing library are allowing this feature.

Moreover, these schemes are enabling a complete encapsulation of these kernels into few subroutines that are stored into a numerical library. By providing an optimized version of this library on each architecture, the code is fully portable without any loss of performance. This point is very important as soon as we want to release the code to physicists. These people are not only using the code as a black box but they are usually introducing some new physical models that are adapted to their particular physical configuration. The concept of the numerical library which is providing the kernels for the computation of the spatial derivatives makes the code as easy to understand as a classical vector code. No extra knowledge is needed especially in the field of the programming of MPP computers. The only point which is actually open is the problem of the I/O since no efficient parallel I/O systems are provided on modern MPP. Figure 1 is showing the global structure of the code.

Even if with distributed memory MPP, the available amount of memory is larger (several Gbytes) than on classical parallel-vector systems, memory usage is remaining a very crucial point in a 3D DNS code. For instance, the computation of the 3D Reynolds stress tensor with variable viscosity and of its derivatives is asking a minimum of 30 3D arrays.

The complete solution of Navier-Stokes equations with a 3 step Runge-Kutta scheme is requiring more than 70 arrays. A plane by plane computation is allowing a reduction of the need of memory by a factor of 4. Only Z derivatives are computed on the whole 3D domain. All other terms are successively computed in 2D XY planes. Moreover, a special treatment of the different terms is allowing a maximum reuse of 3D workspace. But, this method is increasing the amount of messages between the domains since XY derivative kernels are called for each XY plane. Mixing the two previously stated domain decomposition methods in the different directions of the mesh is solving the problem: in Z direction, the low overhead method is used and the low communication one is used in X and Y. By doing so, we have reduced the need of memory without interfering with parallelism and without any degradation of performance. In the final code, we are able to compute an 128^3 point grid in less than 50 Mwords of memory. A memory manager is providing 3D arrays on request and is able to swap-out automatically some arrays on external storage with a LRU scheme or according to the history of the previous requests during previous identical iterations.

In order to be able to compute larger domains at lower cost, different physics are allowed in the different subdomains. Different terms in the equations are activated only when useful. This technique is allowing some gain in memory usage while memory is dynamically allocated depending of the different physical models that are activated in the domain. When coupled to load balancing tool, a large gain in CPU is achievable.

3. Multiphase Computations

An additional physical difficulty is that most DNS of turbulent combustion have been performed using gaseous reactants. DNS of reacting flows with liquid fuel injection is a field in which very little effort has been produced despite the fact that most practical systems use liquid fuels. One of the main reasons for this is that a liquid fuel configuration requires the separate treatment of the gaseous equations and of the droplet movement and evaporation. The problem which has to be considered then is typical of multiphase problems in which two separate physical problems (with different numerical techniques) have to be solved simultaneously and coupled. We have added to our DNS code a 3D droplet solver which is tidily coupled to the gaseous solver. The different droplets are distributed among the different nodes and are belonging to the node which is owning the closest mesh point. Thus, communications are not needed during the coupling of the two models. But, the droplet density is not uniform in the whole domain and is evolving with time. It will have a large influence on load balancing.

4. Load Balancing

Load balancing is remaining a key topic for the efficiency of the whole parallel code. A static load balancing technique is implemented in order to take into account the various costs of boundary condition treatments, the difference in the physic between the domains and in the computational performance of the different nodes. The load balance is determined by the minimization of a cost function with a two-stage simulated annealing method. Even if for most of the computations, this static load balancing is sufficient to assume a good load average between the different processors, a dynamic load balancing

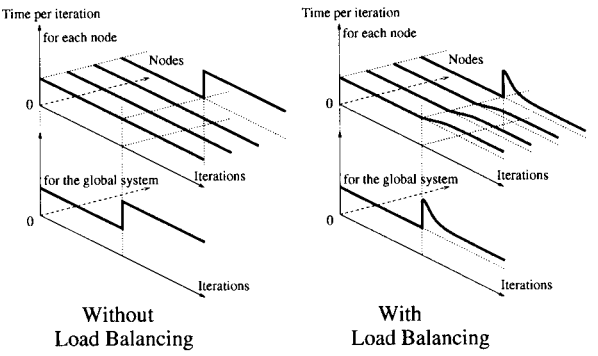


Figure 2. Theoretical behaviour of a non balanced system (left side) and of a dynamically balanced system (right side) when the load on a node is increasing suddently

(DLB) paradigm is needed when running on heteregeneous network in timesharing mode. Averaged load on the different computers is taken into account for the computation of the cost function and the partitionning is dynamicaly modified in order to optimize the efficiency of the computation. The cost function is minimized with a parallel multi-shot simulated annealing algorithm. Figures 2 is showing the theoric behaviour of a DLB method when the load of one node is suddently increasing. Figures 3 is showing the behaviour of the NTMIX3D code when DLB is activated or not. The cost of DLB has been minimized and is less of 2% of the overall cost, but the gain can be over 100% [8]

5. Release to Physicists and Real Computations

After an intensive validation phase, the code has been released to physicist users in the scope of the Centre de Recherche en Combustion Turbulente which is grouping some teams from Institut Français du Pétrole, Ecole Centrale de Paris and CERFACS in Toulouse. This tool is allowing the start of some real studies of 3D turbulent combustion phenomena like turbulence-flame interaction, flame-wall interaction and spark ignition.

Figures 5 and 4 are showing a 161^3 point grid computation of an ignition phenomenon. The flow is initially a homogeneous isotropic turbulent field of premixed gases and the flame is initiated by an heat source term which is representing the heat release of a spark ignition in a reciprocating engine.

6. Summary and Future work

NTMIX3D has been built to study 3D turbulent combustion phenomena. The use of some ad'hoc domain decomposition methods is allowing high efficiency on a wide range of parallel computers as well as of classical shared memory systems. The addition of a memory manager and of a dynamic load balancing tool is enabling a very large flexibility in the use and in the future evolution of the code. We will work now more intensively on multiphysic and on multiphase computations in order to be able to handle new kinds of computations.

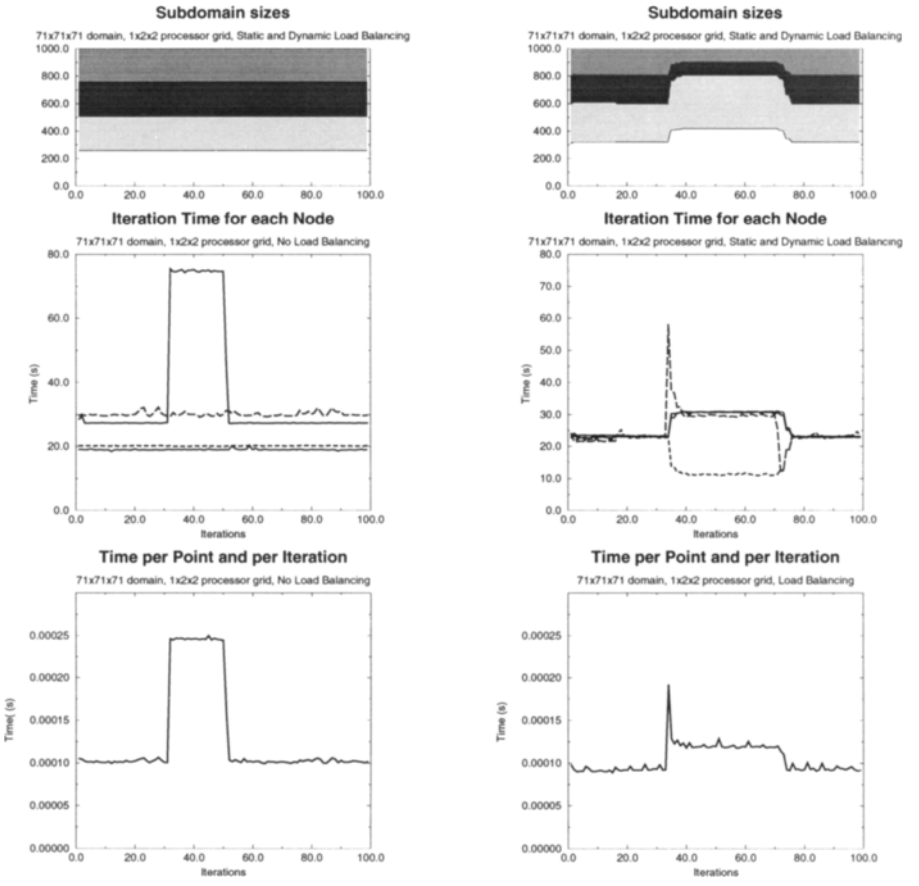


Figure 3. Experimental behaviour of a dynamically balanced system on a 4 processor machine. Static Load Balancing is allowing a perfect initial balance. The balance is not perfect during the perturbation phase because the partitionning is a cartesian one

REFERENCES

1. T. Poinsot, D. Veynante, and S. Candel, "Quenching processes and premixed turbulent combustion diagrams", *J. Fluid Mech*, vol. 228, pp. 561–606, 1991.
2. D. C. Haworth and T. J. Poinsot, "Numerical simulations of lewis number effects in turbulent premixed flames", *J. Fluid Mech*, vol. 244, pp. 405–436, 1992.
3. K. N. C. Bray and R. S. Cant, " Some applications of Kolmogorov's turbulence research in the field of combustion ", *Proc. R. Soc. Lond. A*, vol. 434, pp. 217–240, 1991.
4. S.K. Lele, "Compact finite difference schemes with spectral-like resolution", *J. Comp. Phys.*, vol. 103, pp. 16–42, 1992.
5. T. Poinsot and S.K. Lele, "Boundary conditions for direct simulations of compressible

viscous flows", *J. Comp. Phys.*, vol. 101, pp. 104–129, 1992.

6. A. Stoessel, M. Hilka, and M. Baum, "2D direct numerical simulation of turbulent combustion on massively parallel processing platforms", in *Proc. of the 1994 EUROSIM Conference on Massively Parallel Processing*, Delft (NL), 1994, pp. 783–800, Elsevier Science B. V.
7. A. Stoessel and M. Baum, "Direct numerical simulation of 2D turbulent combustion using domain decomposition methods", in *Proc. of the High Performance Computing 1994 conference*, La Jolla (CA-USA), 1994.
8. J.B. Malezieux and A. Stoessel, "Dynamic load balancing in a direct numerical simulation code of 3D turbulent reactive flow", 1995, Accepted to PARCO 95 conference.

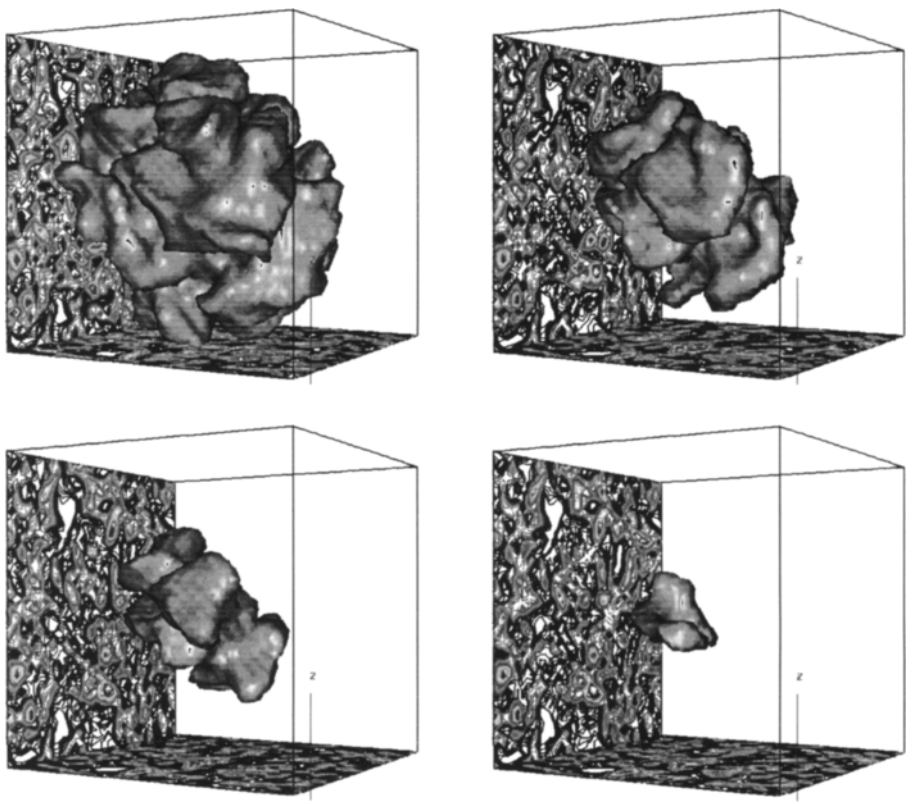


Figure 4. Comparison of flame structure for different Lewis numbers (Lewis=0.4,0.6,0.8,1.0) at a given time. The computations have been achieved with a 161^3 point grid on a 4 processor Fujitsu VPP-500

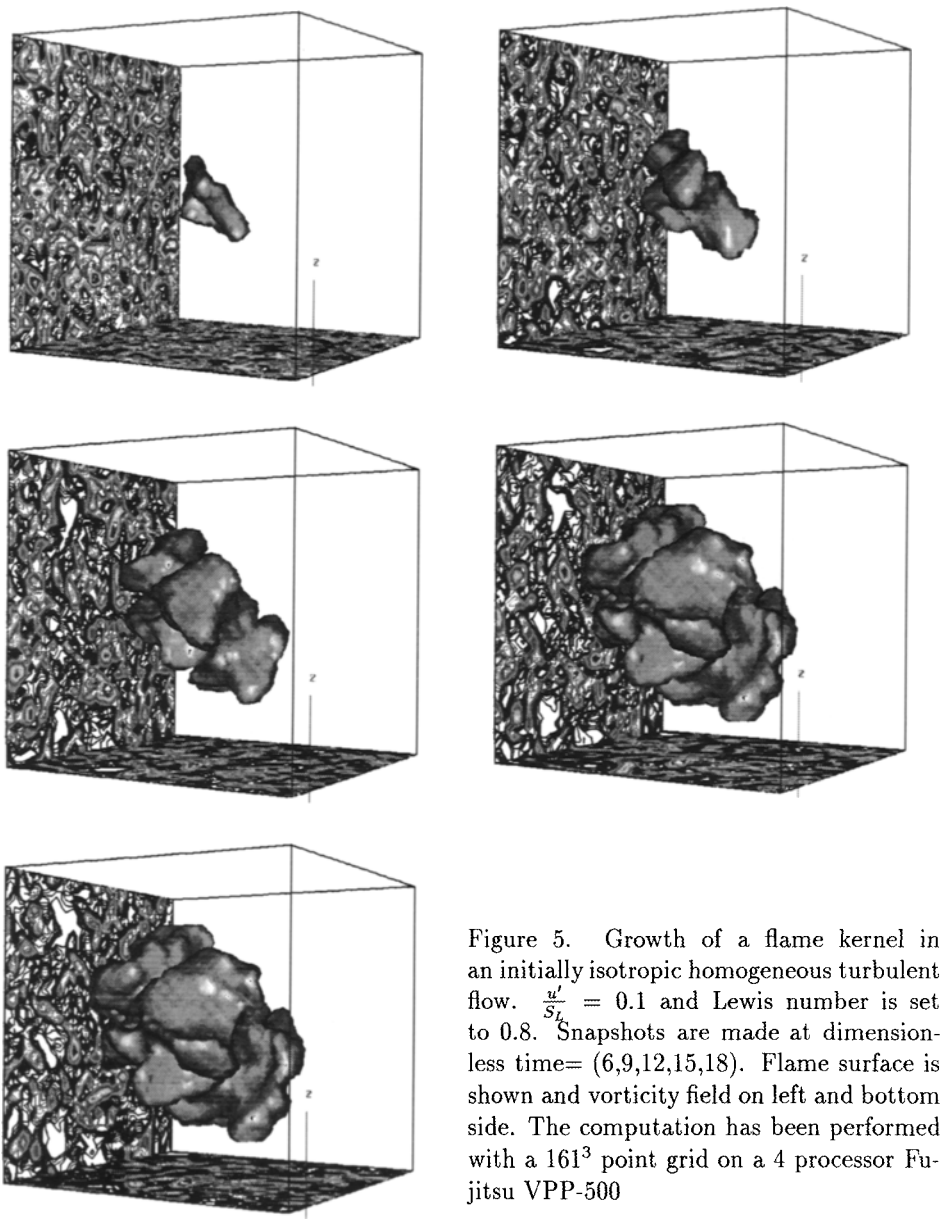


Figure 5. Growth of a flame kernel in an initially isotropic homogeneous turbulent flow. $\frac{u'}{S_L} = 0.1$ and Lewis number is set to 0.8. Snapshots are made at dimensionless time= (6,9,12,15,18). Flame surface is shown and vorticity field on left and bottom side. The computation has been performed with a 161^3 point grid on a 4 processor Fujitsu VPP-500

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Numerical Simulation of Reacting Mixing Layer with Combined Parallel Approach

Edgard Kessy, Alexei Stoukov and Dany Vandromme [a]

[a]LMFN, INSA of Rouen, URA-CNRS 230-CORIA, 76821 Mont Saint Aignan CEDEX, France

This work concerns the parallelization of an explicit algorithm for the simulation of compressible reacting gas flows, applied to supersonic mixing layers.

The reactive Navier-Stokes equations are characterized by three tightly coupled physical phenomena, i.e. the convection, diffusion and chemical source terms. To compute the chemical source terms, full complex chemistry is used. By considering the elapsed time for solving the problem, the numerical treatment of the chemical source terms takes about 75% of the total execution time.

The main goal of the present work is to reduce the relative cost of chemical source terms calculation and also to optimize the global cost of the procedure resolution by the use of parallel computation.

1. Choice of Parallel Approach

On parallel architectures, computations are executed simultaneously on several processors. To achieve this goal, the algorithm is divided into several independent tasks. All tasks can be executed simultaneously and communicate with each other during the execution.

Two different types of parallel methodologies exist: data-parallelism and control parallelism [14]. The first approach relies on the fact that each processor performs at a given time the same instruction on different data. This approach exploits the great potential of massively parallel computers, SIMD (Single Instruction Multiple Data) architectures [11].

In the control-parallelism approach, the computational problem is divided into a number of independent tasks, with different processors performing different tasks in parallel. This approach is adapted to multi-processor computers, MIMD (Multiple Instruction Multiple Data) architectures [1].

In order to use MIMD computers very efficiently, the granularity of tasks should be as large as possible. In CFD the decomposition of the computational domain is the most efficient technique in order to increase the granularity of tasks for MIMD architectures [5]. This technique (the so-called multi-domain technique) consists in partitioning the computational domain into a number of blocks, and to distribute blocks onto different processors [4]. However, there is still a problem to maintain a well balanced decomposition.

The choice of the adopted parallel approach is influenced by the numerical algorithm.

However, one can notice that data-parallel and control-parallel approaches can both be combined [12]. By considering that in most reacting flows, reacting and non-reacting zones occur simultaneously, the computation of chemical source terms can be restricted to the reactive region. Thus a decomposition efficient for pure hydro dynamical problem becomes inefficient when the reacting zone dimensions differ greatly between blocks. In such a way, the standard multi-block technique is no longer well suited for the reacting flow. In this paper, a multi-block technique is used for convective and diffusive terms, whereas SPMD (Single Program Multiple Data) or SIMD approach is employed for chemical source terms.

2. Flow configuration, physical, chemical and mathematical model

Figure 1 shows the physical model considered in the present study. It consists of two chemically active hydrogen and air streams with different stream wise velocities. The spatial mixing of reacting streams has been simulated in a two-dimensional domain.

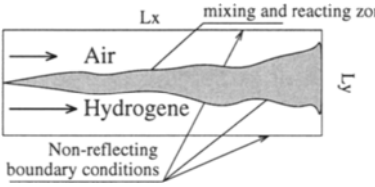


Figure 1. Flow configuration

The static pressure at the inlet side is the same for both streams. To prescribe the inlet conditions the self similar solution of the compressible mixing layer [9] is used at the inlet with a vorticity thickness equal to 0.05 of the transverse length. The inlet conditions are presented in Table 1.

Table 1

Stream	Velocity m/s	Mach number	Pressure Pa	Temperature K	Convective Mach number
H_2	3000	1.24	$1.013 \cdot 10^5$	$930 - 1000$	0.446
AIR	1780	2.803			

The flow evolution is governed by the unsteady compressible Navier-Stokes equations coupled with the energy and species transport equations. These equations are written in two-dimensional form as:

$$\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} + \frac{\partial G(U)}{\partial y} = S(U) \tag{1}$$

where the variable vector, the convective and diffusive fluxes and source terms are defined respectively by:

$$U = [\rho_1, \cdots, \rho_{nsp}, \rho u, \rho v, \rho e_t]^T$$

$$F = F_c + F_v \qquad G = G_c + G_v$$

$$F_c = [\rho_1 u, \cdots, \rho_{nsp} u, \rho uu + P, \rho uv, (\rho e_t + p)u]^T$$

$$F_v = -\left[\sigma_{Y_1 x}, \cdots, \sigma_{Y_{nsp} x}, \sigma_{xx}, \sigma_{xy}, \sigma_{xx} u + \sigma_{xy} v - q_x\right]^T$$

$$G_c = [\rho_1 v, \cdots, \rho_{nsp} v, \rho vu, \rho vv + p, (\rho e_t + p)v]^T$$

$$G_v = \left[\sigma_{Y_1 y}, \cdots, \sigma_{Y_{nsp} y}, \sigma_{xy}, \sigma_{yy}, \sigma_{xy}u + \sigma_{yy}v - q_y\right]^T$$

$$S = \left[\dot{\rho}, \cdots, \dot{\rho}_{nsp}, 0, 0, \sum_{j=1}^{N_r} Q_{r_j}\dot{w}_{r_j}\right]^T$$

$$\sigma_{Y_k x} = D_k \frac{\partial Y_k}{\partial x} : w \qquad \sigma_{Y_k y} = D_k \frac{\partial Y_k}{\partial y} \qquad \sigma_{xx} = \mu(\frac{4}{3}\frac{\partial u}{\partial x} - \frac{2}{3}\frac{\partial v}{\partial y}) \qquad \sigma_{yy} = \mu(\frac{4}{3}\frac{\partial v}{\partial y} - \frac{2}{3}\frac{\partial u}{\partial x})$$

$$\sigma_{xy} = \mu(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}) \qquad q_x = -\lambda\frac{\partial T}{\partial x} - \sum_{k=1}^{nsp} h_k \sigma_{Y_k x} \qquad q_y = -\lambda\frac{\partial T}{\partial y} - \sum_{k=1}^{nsp} h_k \sigma_{Y_k y}$$

The chemical kinetic equations are represented by:

$$\sum_{k=1}^{nsp} a_{kr} X_{kr} \Longleftrightarrow \sum_{k=1}^{nsp} b_{kr} X_{kr} \tag{2}$$

where X_{kr} represents one mole of species k, a_{kr} and b_{kr} are integral stoichiometric coefficients for reaction r. The rate of changes due to r^{th} chemical reaction is

$$\dot{w}_r = K_{fr}\prod_k (\frac{\rho_k}{W_k})^{a_{kr}} - K_{br}\prod_k (\frac{\rho_m}{W_k})^{b_{kr}} \tag{3}$$

where K_{fr} and K_{br} come from a generalized Arrhenius form.

In this work the hydrogen-air combustion kinetics is taken as the detailed mechanism proposed in [7]. The concentrations of H_2, O_2, OH, H_2O, H, O, HO_2, H_2O_2 and N_2 species evolve according to a 37 chemical reactions scheme.

To describe the complex properties of molecular diffusion and thermal conduction the appropriate modules of CHEMKIN-II package [6] were incorporated into the code. The values of enthalpies of formation and specific heats of species are obtained with the JANAF tables [13].

Spatial instability of hydrogen-air mixing zone is generated by disturbing the inlet velocity profile. In this work, the initially incompressible disturbances are imposed only on the longitudinal velocity component at the point of maximum of inlet vorticity profile. The disturbances introduced are $U' = A\cos(\omega t)$, where A is the magnitude equal to 0.5% of the inlet velocity taken at the point of the maximum vorticity. The ω is the circular frequency of oscillation and is chosen from the linear stability analysis results [9].

3. Numerical Algorithm

The equation (1) is solved with fractional-step approach [15]. Let us denote δU_c, δU_v and δS the increments of vector U which are due to convection, viscosity and the chemical source terms respectively. The overall increment of U is then obtained as their sum:

$$\delta U = \delta U_c + \delta U_v + \delta S \tag{4}$$

The numerical scheme is written here for one-dimensional in-viscid part of equation (1). Following [16,17,8] the numerical solution for second-order upwind TVD (total variation diminishing) scheme is :

$$\delta U^n_{c_i} = -\lambda(F^n_{c_{i+1/2}} - F^n_{c_{i-1/2}}) \tag{5}$$

where $\lambda = \Delta t/\Delta x$, $F_{c_{i\pm1/2}}$ is a numerical flux vector, which is expressed for a non-MUSCL approach as

$$F_{c_{i+1/2}} = \frac{1}{2}(F_{c_i} + F_{c_{i+1}} + R_{i+1/2}\Phi_{i+1/2}) \tag{6}$$

The increment of vector U due to viscous/diffusion fluxes is obtained by the central difference approximation:

$$\delta U_{v_i} = \frac{\lambda}{2}\left(F_{v_{i+1/2}} - F_{v_{i-1/2}}\right) \tag{7}$$

where, for example for species k

$$F_{v_{i+1/2}} = D_{k_{i+1/2}} \frac{Y_{k_{i+1}} - Y_{k_i}}{\Delta x_{i+1/2}} \tag{8}$$

To compute the δS three different algorithms (CHEM [2], EULSIM [3], LSODE [6]) have been tested. The most accurate results have been obtained with EULSIM scheme which is, however, the most demanding of CPU time.

Second-order accuracy for the time integration is obtained using a second-order Runge-Kutta method.

4. Parallelization

For the splitting of a computational domain into N blocks having the same number of mesh nodes, the calculations are performed according equation (4).

4.1. Hydrodynamic part parallelization

The increments $\delta U_{c_i,j}$ and $\delta U_{v_i,j}$ are found with the standard multi-blocks technique, where each processor unit treats a single block. Here the computational domain may be splitted in arbitrarily, the decomposition being dependent of the particular flow geometry. Numerical scheme employs five-point discretisation. Therefore to find $\delta U_{c_i,j}$ at i-th point, the values at points set (i-2,i-1,i,i+1,i+2) are needed. It means that every neighboring blocks should have four common mesh nodes. An example of such decomposition is shown on Fig.2 for two adjacent blocks.

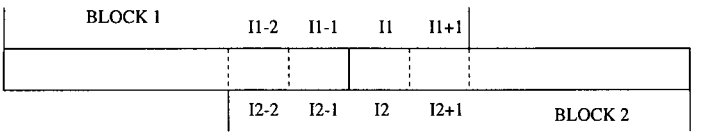


Figure 2. Linking up of the neighboring blocks

Having computed $\delta U_{c_i,j}$ and $\delta U_{v_i,j}$, the neighboring blocks 1 and 2 are linked up in a following manner:

the columns $I1-2$ and $I1-1$ of the block 1 become those $I2-2$ and $I2-1$ of the block 2;

the columns $I2$ and $I2+1$ of the block 2 become those $I1$ and $I1+1$ of the block 1.

In such a manner we need only the communications between the processors treating the adjacent blocks.

4.2. Parallelization of Chemical Source Terms Computation

To reduce the computer time, the chemical source terms are calculated only on the mesh nodes where the chemical reactions take place in "reactive" regions. Size of that region may differ greatly from one block to another, so the domain decomposition with equal number of grid points may be highly ineffective. This is the case, for example, for the simulations of reactive mixing layer, triple flames, jets etc. Evidently the overall computations rate (and cost) will be determined with the time needed to treat a block with the largest reactive sub-zone. Thus we need to equalize the number of "reacting" points for each processor.

The MIMD-SPMD algorithm may be represented as:

Slaves:

- to determine the reactive region
- to rearrange the data as the vectors
- to send to master a number of points in which the chemical source terms should be calculated.

Master:

- to get these numbers for each blocks
- to determine the relative load of each slave and the numbers of data points to be exchanged between them
- to send the orders for data transfer.

Slaves:

- to get the master's orders and to perform corresponding transactions: to get the additional scalar values sets for unloaded processor-slave or to send them for overloaded one. If the load is near the mean one there is no data transfer.
- to compute δS
- to send back the modified scalar values to the corresponding slave
- to rearrange the data back to the spatial distributions.

The second approach used here is both MIMD-SIMD type. The source terms solver have been adapted for MASPAR computer (DPU MP1024 with 4096 processors layed on a 64x64 grid) [14]. In a such architecture the MASPAR front-end plays the master role and 5 IBM RS6000K are slaves.

The numerical procedure can be described in term of tasks as:

Master:

- collection of data required for the chemical source terms calculation from the whole set of slaves;
- rearrangement of there data in 64x64 grids;
- dispatching of the resulting data towards the MASPAR DPU;
- computation of source terms on DPU;
- compilation of the data resulting from the source terms calculation;
- inverse rearrangement of the data, before sending towards the slaves.

Slaves:

- sending of the data concerning the chemical source terms;
- hydrodynamic phenomena calculation;
- collection of chemical source terms results from the master;
- numerical evaluation of $\overline{U_{i,j}^{n+1}}$ (predictor) or $U_{i,j}^{n+1}$ (corrector)

5. Results

All the calculations have been performed with cluster of 5 IBM RS6000K workstations and MASPAR DPU MP1024 computer. The MIMD part relies on the PVM communication library. An effectiveness of the parallelization method has been tested upon the modeling of supersonic mixing layer with and without chemistry.

5.1. Modeling of Inert Mixing Layer

In this case the equal load of each processor is obtained simply with the equal partitioning of the computational domain. Multi-block decomposition linking up on the boundaries do not introduce any numerical disturbance in comparison with the usual single block calculation, no matter how the domain has been splitted. However, for the computational mesh of $N_x * N_y = 200 * 100$ nodes, decomposition along x-axe main flow direction appeared to be much more effective. It can be explained with that resulting blocks of $200/Nbl * 100$ allow to take all the advantages of cache memory. Therefore all the computations have been performed with the decomposition along x-direction.

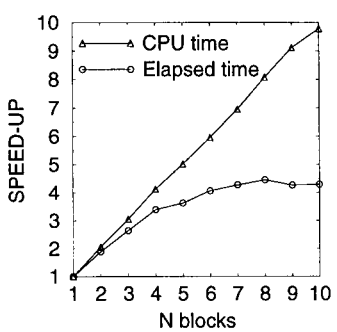


Figure 3. Speed-up in comparison with sequential simulation for inert mixing layer

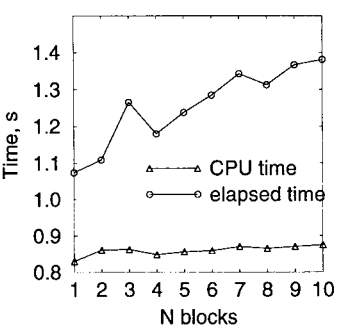


Figure 4. Time increasing: case of constant blocks size

Speed-up is illustrated on Fig.3. The difference between the gain in CPU time and that in elapsed (i.e. real user time) is due to the non-negligible communication time between the processors. Although the time of computations is decreased with successive increases of block number, communications time becomes exceedingly large. As a result we obtain some efficiency threshold in effective user time.

Figure 4 present the plot of time versus the number of same size blocks. In this case the CPU time is still constant, but the elapsed time shows a small growth due to the increasing number of communications.

5.2. Reactive Mixing Layer

As it was mentioned above, chemical source terms are most CPU-time-demanding. Usually they consume more than 75% of overall CPU time. So, any progress in their parallelization is rather crucial, and one can expect that improved efficiency would be more significant than for non-reacting case.

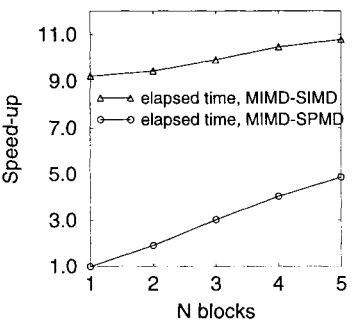


Figure 5. Speed-up in comparison with sequential simulation for reacting mixing layer

On Fig.5 the results are presented for hydrogen-air mixing layer with real complex chemistry. Speed-up in case of MIMD-SPMD algorithm increases linearly with the number of blocks. This tendency may be explained by the fact that the communication time between blocks is negligible in comparison with the computation time. In the case of one block use, the MIMD-SIMD approach shows speed-up of approximatively 9 in comparison with the sequential calculation. Furthermore the number of blocks increases, the speed-up improvement given by MIMD-SIMD algorithm become of the less interest, because both computation time and data manipulation time are the similar.

6. Conclusion

In the present work two combined parallel approach are proposed for the problems with complex reaction zone structure. The performances of these consecutive algorithm has been tested in case of simulation of hydrogen-air supersonic mixing layer. It yields a significant time saving with respect to the ordinary multi-block technique or the sequential calculation.

MIMD-SPMD algorithm is less efficient than the MIMD-SIMD one when the number of blocks is small. Nevertheless this algorithm is still easy to implement. MIMD-SIMD algorithm shows high performances in case of one slave configuration. But numerical implementation of this algorithm requires a typical massively parallel chemical source solver.

REFERENCES

1. R. Aggarwal, P. Henriksen, R. Keunings, D. Vanderstraeten, and O. Zone. Numerical simulation of non-newtonian flow on MIMD parallel computers. In Hirch, editor, *Computational Fluid Dynamics'92*, pages 1139–1146, 1992.
2. A.A. Amsden, P.J. O'Rourke, and T.D. Butler. KIVA-II: A computer program for chemically reactive flows with sprays. Report LA-11560-MS, Los Alamos National Laboratory, Los Alamos, New-Mexico 87545, 1989.
3. P. Deuflhard. Uniqueness theorems for stiff ode initial value problems. Preprint SC-87-3, Konrad-Zuse-Zentrum fuer Informationstechnik Berlin, 1987.
4. St. Doltsinis, I. and S. Nolting. Generation and decomposition of finite element models for parallel computations. *Computing Systems in Engineering*, 2(5/6):427–449, 1991.
5. H. Friz and S. Nolting. Towadrs an integrated parallelization concept in computational fluid dynamics. In *ERCOFTAC Bulletin*, october 1993.
6. R.J. Kee and J.A. Miller. A structured approach to the computational modeling of chemical kinetics and molecular transport in flowing systems. SAND86-8841, Sandia National Laboratories, 1986.
7. U. Maas and J. Warnatz. Ignition processes in hydrogen-oxygen mixtures. *Combustion and Flame*, (74), 1988.
8. J.L. Montagne, H.C. Yee, and Vinokur M. Comparative study of high-resolution shock-capturing schemes for a real gas. *AIAA Journal*, 27(10):1332–1346, 1989.
9. O.H. Planche and W.C. Reynolds. A numerical investigation of the compressible reacting mixing layer. Report TF-56, Stanford University, Stanford, California 94305, October 1992.
10. P.L. Roe. Some contribution to the modelling of discontinuous flows. In American Mathematical Society, editor, *Lectures in Applied Mathematics*, pages 163–194, Providence, RI, 1985.
11. M.L. Sawley. Control- and data-parallel methodologies for flow calculations. In *Supercomputing Europe'93*, Ultrecht, February 1993. Royal Dutch Fairs.
12. M.L. Sawley and J.K. Tegner. A data parallel approach to multiblock flow computations. *International Journal For Numerical Methods in Fluids*, 19:707–721, 1994.
13. D.R. Stull and Prophet H. JANAF thermochemical tables, second edition. NSRDS-NBS 37, U.S. Department of Commerce/National Bureau of Standards, June 1971.
14. D. Vandromme, L. Vervisch, J. Reveillon, Y. Escaig, and T. Yesse. Parallel treatment of CFD related problems. In preparation, 1994.
15. N. N. Yanenko. *The method of fractional steps.* New York, Springer Verlag edition, 1971.
16. H.C. Yee. Construction of explicit and implicit symmetric TVD schemes and their applications. *Journal of Computational Physics*, 68:151–179, 1987.
17. H.C. Yee. A class of high-resolution explicit and implicit shock-capturing methods. Computational Fluid Dynamics, March 1989, Rhode-St-Genèse, Belgium, Von Karman Institute for Fluid Dynamics, Lecture Series 1989-04, 1989.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

On the optical properties of a supersonic mixing layer

Yeong-Pei Tsai

Department of Mechanical Engineering, Chung-Hua Polytechnic Institute, Hsin Chu, Taiwan, ROC.

Physical properties of spatially developing shear layers between two flows of different velocities are investigated. The two fluid streams are supersonic and the convective Mach number, M_c, is greater than unity in most cases. The two-dimensional compressible Euler equations are solved directly using the explicit Godunov method. A hyperbolic-tangent velocity profile is adopted for the initial streamwise velocity distributions at the splitter plate. Temporal and optical statistics of the mixing layer are calculated. The optical properties of the mixing layer are determined by computing the Strehl ratio. Although the growth rate of a supersonic mixing layer is much smaller than that of a subsonic mixing layer, it is found that the far-field peak intensity is less than that due to a subsonic mixing layer at the same velocity ratio and density ratio.

1. BACKGROUND AND PURPOSES

The spatially-developing, plane free shear layer generated by the turbulent mixing of two co-flowing fluid streams is gemetrically simple (Fig. 1(a)). However, this simple flow configuration is important in mixing processes and is encountered in many other engineering applications. So the fundamental properties of plane mixing layer have been the subject of extensive investigations for many years. Since there are many circumstances which involve the interaction between turbulent mixing layers and laser beam [1], one of the current research topics in this area is the optical properties of shear layer.

The extraction of power from high-power gas laser, for example, often involves passing the beam through interface between gases of different indices of refraction (e.g. aerodynamic window). Turbulent shear layers can produce random optical phase errors in the beam that can substantially reduce the maximum intensity to which the beam can be focused. Similar examples involving beam degradation are found in cases like aero-optics and atmospheric propagation of light. In the past, the investigation of shear layer optical properties were based on the assumptions that the natural shear layers were homogeneous and isotropic [2, 3]. There has been no study of the

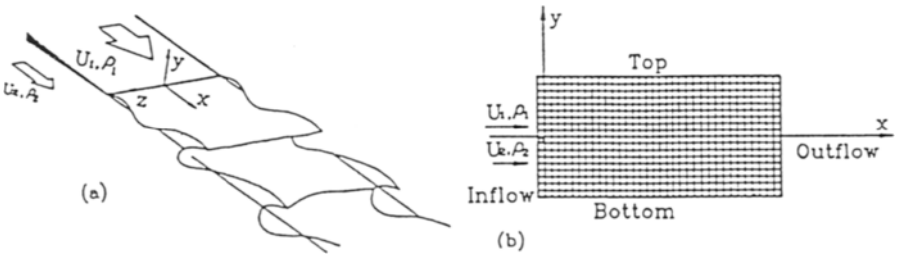


Fig.1 (a) Flow configuration and geometry. (b) Schematic diagram of the computational domain.

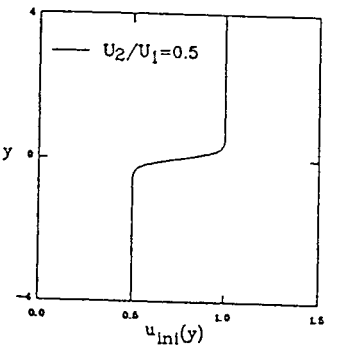


Fig.2 The initial streamwise velocity distribution: a hyperbolic–tangent velocity profile

of the optical effects of large-scale structures which are existing during the course of shear layer development. A systematic numerical simulation using the methods of computational fulid dynamics in this field has carried out for subsonic mixing layer [4]. The optical effects of coherent structures in the mixing layer were identified and the methods of improving optical performance in the far-field by effective controlling of the mixing layer were found. Since the aerodynamic window may be supersonic, the optical effects of a supersonic mixing layer are explored in this paper. The compressibility effects of the supersonic flow are emphasized.

2. NUMERICAL METHODS

The two-dimensional compressible Euler equations are solved directly using the explicit Godunov methods. A hyperbolic-tangent velocity profile is adopted for the initial streamwise velocity distributions at the splitter plate (Fig. 2). Fig. 1a indicates the flow configuration and geometry where **x** is the streamwise coordinate and **y** is the cross-stream coordinate. The upper flow velocity, U_1, is always larger than the lower one, namely U_2. Fig. 1b is the schematic diagram of the computational domain. A grid of 200 by 50 is adopted. The computational cell is a square mesh with side length 0.02cm. The detailed numerical algorithm can be found in Ref. 4.

3. RESULTS AND DISCUSSIONS

3.1. Fluid mechanical results

As a typical example, the instantaneous flow visualization of the density field for the natural shear layer with $M_1=4$ and $M_c=1.024$ is illustrated in Fig. 3 which shows that supersonic shear layer also exhibits well-defined coherent structures, though not as organized as those found in a subsonic mixing layer [4, 5]. It is obvious that the spreading rate of the shear layer is unusually small for supersonic mixing layer. The convective Mach number M_c is a useful parameter describing supersonic flow which is defined as

$$M_c=\frac{M_1(1-\lambda_u)}{[1+\lambda_\rho^{-0.5}]},$$

where $\lambda_u=\frac{U_2}{U_1}$ is the velocity ratio and $\lambda_\rho=\frac{\rho_2}{\rho_1}$ is the density ratio. The main result shown in Fig. 4, is the plot of nondimensional mean streamwise velocity, $\bar{\zeta}_u(x,y)$, versus the

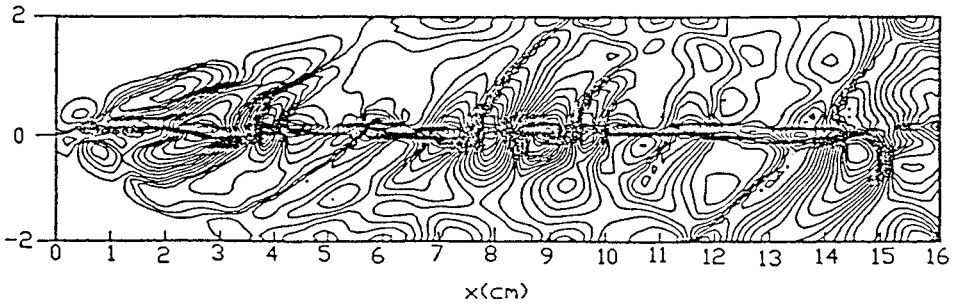


Fig.3 Instantaneous isodensity plot for M_1=4, M_c=1.024.

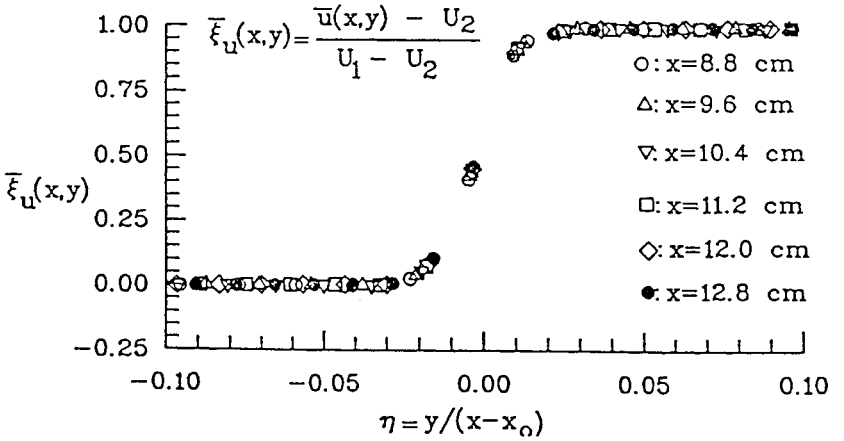


Fig.4 The mean streamwise velocity profile; M_1=4, M_c=1.024.

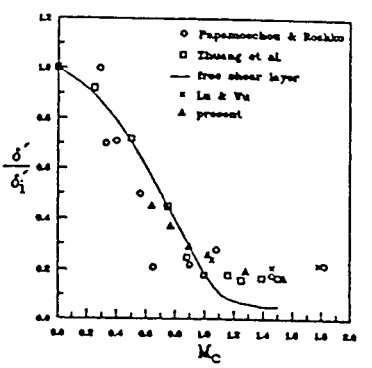


Fig.5 A comparison among computed, theoretical, and experimental normalized growth rates.

similarity coordinate η, where $\xi_u(x,y)=\frac{\bar{u}(x,y)-U_2}{U_1-U_2}$ and $\eta=\frac{y}{x-x_o}$. The lateral distribution of the mean streamwise velocity suggests that the supersonic free shear layer is also self-similar in the fully-developed region. Assuming that the extent of the mixing layer is bounded and which is the region with ξ_u in the range of 0.01 to 0.99, the local width of the free shear layer is defined by $\delta(x) = y_{0.99} - y_{0.01}$ at a fixed streamwise location **x**, where $y_{0.01}$ is the **y**-location at which $\bar{u} = U_2 + 0.01\Delta U$ and $y_{0.99}$ is the location at which $\bar{u} = U_2 + 0.99\Delta U$. Then the growth rate of the supersonic mixing layer, $\frac{d\delta}{dx}$, can be calculated according to this definition. The effect of convective Mach number, M_c, on shear layer thickness is uncoupled by comparing the compressible growth rate to the incompressible growth rate at the same density and velocity ratio. This is made by plotting $\frac{d\delta}{dx}/(\frac{d\delta}{dx})_i$ against M_c and this is shown in Fig. 5, where $(\frac{d\delta}{dx})_i$ is the incompressible growth rate at the same values of λ_u and λ_ρ as that for the supersonic mixing layer. In Fig. 5, the results are also compared with the experimental results of Papamoschou and Roshko [6], with the results of Zhuang et al. which is calculated according to the inviscid instability theory [7], with the results of Lu and Wu [8], and with the results obtained from the 2-D free shear layer calculations [9]. Since in the present calculations, the boundary conditions used are based on the assumption that the top and the bottom boundaries are considered as streamlines in the computatioal domain, the results may be classified as those correspondig to a bounded shear layer. It is obvious that the growth rate of the 2-D supersonic shear layer approaches an asymptotic value as the convective Mach number becomes supersonic, which is consistent with the experimental results and values.

The time averaged density contours in Fig. 6 illustrate the interaction between the reflected shoch wave by the wall and the shear layer, where M_1=4 and M_c=1.024. Since the density ratio is 1.1, this shear is corresponding to an overexpanded case in accord with Guirguis et al. [10]. Also, it is noticed that in the mixing layer there is a shock/shear interaction region which can be identified around **x**=11cm in this case. However the change in growth rate is not significant there. Samimy and Elliot [11] showed the similar result. The trends of the computed results discussed above to experimental and numerical results of others lend full confidence that the code has been validated satisfactorily for the supersonic mixing layer and the density field may be used for optical properties study and simulation.

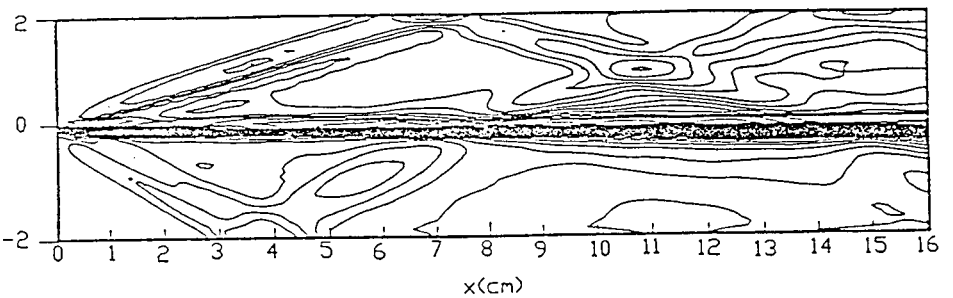


Fig.6 Time averaged isodensity plot for M_1=4, M_c=1.024.

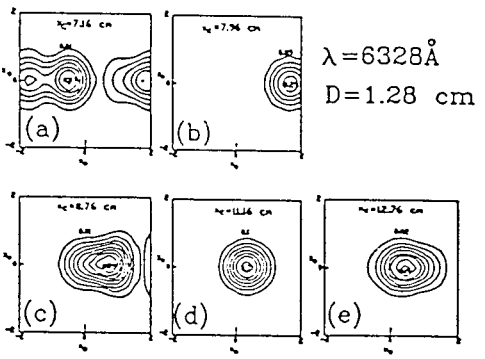


Fig.7 Instantaneous far–field intensity distribution at different locations of the beam center M_1=4, M_c=1.024.

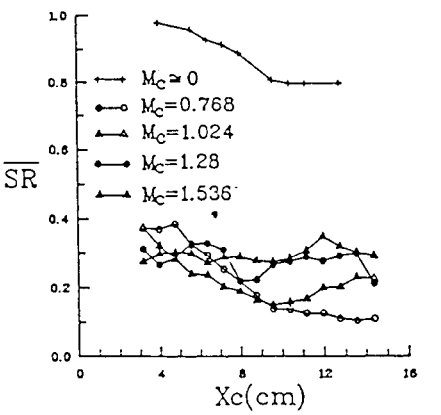


Fig.8 Time averaged Strehl ratio at different Convective Mach numbers

3.2. Shear layer optical statistics

The optical effects of the shear layer are calculated by passing a laser beam through it with circular aperture and uniform phase. The far-field focal plane intensity distributions measure the optical quality of the shear layer. The Strehl ratio, SR, which is defined as the ratio of the maximum light intensity of the diffraction pattern to that of the same optical system without aberration, will be used to evaluate the optical quality quantatively.

A series plots of the instantaneous far-field intensity contour due to the supersonic shear layer with M_1=4 and M_c=1.024 are shown in Fig. 7. The beam size is D=1.28cm (the diameter of the beam) and $\mathbf{x}_c$ is the location of the center of the beam. In Fig. 7a, the scale of the fluid fluctuations is smaller than the beam size, and there are structures, i.e. the lobes formed in the scattered far-field intensity pattern because the beam contains more than one large eddies. As the center of the beam moves further downstream where the size of the flow disturbance is a fraction of the beam size, and then the shape of intensity distribution is only distored. The shift in location of the peak intensity in the far- field is due to the tilt effect of the shear layer.

The effect of convective Mach number, M_c, on the Strehl ratio is illustrated in Fig. 8 where the time averaged Strehl ratio, $\overline{SR}$, is plotted as function of $\mathbf{x}_c$ with density ratio 1.1, D=1.28 cm and the wavelength of beam is 6328A. For smaller M_c, the spreading rate of the mixing layer is larger. This implies a thicker mixing layer at a fixed streamwise location for smaller convective Mach numbers. Therefore the amplitudes of optical phase fluctuations are enhanced there, and the beam degradation is larger for the mixing layer at that position. However, in the upstream region of the shear layer, it is noted that the difference in Strehl ratio is not significant for different M_c.

4. CONCLUSION

A comparison between the values of Strehl ratio due to supersonic shear layer and a subsonic one with the same velocity ratio and density ratio is also made (Fig. 8). Although the growth rate of supersonic shear layer is unusually small, the Strehl ratio is not high as expected because the shock/shear layer interaction region enhances the optical random phase errors in the beam passing through the shear layer that can substantially reduce the maximum intensity to which the beam can be focused in the far-field. Unless a method for the optical phases compensation is available, a supersonic aerodynamic window is not recommended in a high-power gas laser system.

REFERENCES

1. Craig, J. E., and Aller, C., " Aero-optical turbulent boundary layer/shear layer experiment on the KC - 135 aircraft revisited, " Optical Engineering, Vol. 24, No. 3, 1985, pp. 446 - 454.
2. Vu, B. T., Sutton, G. W., Theophanis, G., and Limpaecher, R., " Laser beam degradation through optical turbulent mixing layers, " AIAA paper 80-1414, 1980.
3. Sutton, G. W., " Effect of turbulent fluctuations in a optically active medium, " AIAA Journal, Vol. 7, 1969, pp. 1737 - 1743.
4. Tsai, Y. P., and Christiansen, W. H., " Two-dimensional numerical simulation of shear layer optics, " AIAA Journal, Vol. 28, No. 12, 1990, pp. 2092 - 2097.
5. Brown, G. L., and Roshko, A., " On density effects and large structures in turbulent mixing layer, " Journal of Fluid Mechanics, Vol. 64, June 1974, pp. 775 - 816.
6. Papamoschou, D., and Roshko, A., " Observation of supersonic free shear layers, " AIAA Paper 86 - 0162, Jan., 1986.
7. Zhuang, M., Dimotakis, P. E., and Kubota, T., " The effect of walls on a spatially growing supersonic shear layer, " Phys. Fluids A, Vol. 2, No, 4, 1990, pp. 599 - 604.
8. Lu, P. J., and Wu, K. C., " Numerical investigation on the structure of a confined supersonic mixing layer, " Phys. Fluids A, Vol. 3, No. 12, 1991, pp. 3063 - 3079.
9. Zhuang, M., Kubota, T., and Dimotakis, P. E., AIAA Paper 88 - 3588 - CP, 1988.
10. Guirguis, R. H., Grinstein, F. F., Young, T. R., Oran, E. S., Kailasanath, K., and Boris, J. P., AIAA Paper 87 - 0373, 1987.
11. Samimy, M., and Elliot, G. S., " Effects of compressibility on the characteristic of free shear layers, " AIAA Journal., Vol. 28, No. 3, 1990, pp.439 - 445.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Computation of chemically reacting flow on parallel systems

J.G.Carter[a], D.Cokljat[a] , R.J.Blake[a] and M.J.Westwood[b]

[a]CLRC, Daresbury Laboratory, Warrington WA4 4AD, United Kingdom.

[b]Tioxide Group Services Ltd., Billingham, Cleveland, United Kingdom.

The aim of this paper is to report on progress made in the development of a generally-applicable predictive procedure for chemically reacting turbulent flows in various types of axisymmetric reactors. For that purpose a code, which solves the Navier-Stokes equations for chemical reactor problems, has been developed. The main contribution here is in the assessment of a complete Reynolds-stress transport model (RSM). The RSM was compared with the results obtained by a standard k-ϵ model for the chemical reactor considered in the present study. Both models, together with the governing equations, were discretized using a co-located, finite-volume method. The coupling between the pressure and velocity equations is achieved using the SIMPLE algorithm and the resulting iterative equations are solved using either the standard Tri-Diagonal Matrix Algorithm (TDMA) or a preconditioned conjugate gradient algorithm (PCCG). A parallel version of the code was developed within the Communicating Sequential Process (CSP) programming model, implemented within a message passing harness portable to both shared- and distributed-memory systems. Parallel performances of the code are presented for the Intel i860 and IBM-SP2.

1. INTRODUCTION

It is very often the case that a high level of mixing of reactants is required within a reactor in order to ensure complete reaction. Therefore reactors need to provide the conditions for efficient mixing and this, together with the chemical kinetics, make the flow within a reactor very unstable and more turbulent when compared to non-reacting flows. This is the main motivation for introducing and testing more advanced turbulence models for the chemically reacting flows. More advanced turbulence modelling, improved chemical reaction kinetic schemes and numerically finer grids are prerequisite for better predictions of chemically reacting flows. These higher quality calculations are computationally very demanding and require large scale computational resources which will only be met through the application of parallel processing technology.

1.1. Description of problem

We are concerned with chemically reacting flows in pipe type geometries. The geometries can be modified with the inclusion of obstacles, giving a variety of reactor configurations which can be studied. The particular reaction of interest is the oxidation of titanium tetrachloride to produce pigmentary titania (TiO_2). This oxidation process involves sev-

eral complicated stages which may include the nucleation, growth and coagulation of the titania and the dissociation and recombination of chlorine product. In general the reactors have multiple inlets through which the feed materials (O_2 and $TiCl_4$) are introduced at various temperatures and velocities. The products (and any unreacted feed materials) are convected downstream and out of the reactor.

An important feature of the current numerical scheme is the way the titania pigment is modelled. Initially titania nuclei are produced which, as they proceed down the reactor, grow. The particles will also collide and may coalesce together. This process produces a continuous size distribution of titania particles. Numerically the continuous particle size domain is discretized into a number of discrete size intervals. Details of this process can be found in Hounslow et al. [1]. A schematic of the overall process with a typical reactor configuration is shown in Figure 1.

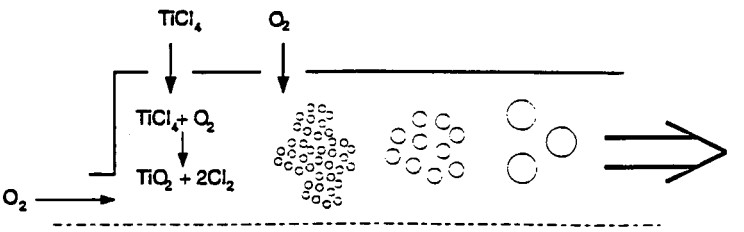


Figure 1. Typical reactor configuration and pigment formation

2. MATHEMATICAL MODEL

The conservation of the mass and momentum, for steady-state variable density flow, may be described by the equations:

$$\frac{\partial(\rho U_i)}{\partial x_i} = 0 \quad ; \quad \frac{\partial(\rho\, U_i\, U_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\mu\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) - \rho\,\overline{u_i u_j}\right] \tag{1}$$

The above set of equations is not closed and in order to achieve closure, initially the standard k-ϵ model of turbulence is considered. In this model, the unknown Reynolds stresses ($\overline{u_i u_j}$) are linearly correlated to the mean rate of strain as follows:

$$-\rho\,\overline{u_i u_j} = \mu_t\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) - \frac{2}{3}\rho\,\delta_{ij}\,k \tag{2}$$

where the turbulent viscosity μ_t is given by: $\mu_t = \rho\, C_\mu\, k^2/\epsilon$. The kinetic energy of turbulence (k) and the dissipation rate of turbulent kinetic energy (ϵ) are determined from their own transport equations.

In the present study, a fast chemistry approach has been assumed where the time scale of chemistry is very small compared with the turbulence time scale (given by k/ϵ).

Consequently it is logical to expect that the reaction rate of the chemical process within the reactor would rather be mixing then chemically dominated. In such cases the accurate predictions of turbulence is essential and therefore we consider the Reynolds-stress closure as an alternative approach to the above mentioned k-ϵ model. The Reynolds stress models, which are known to be more successful in predicting the turbulent flows (e.g. flows with recirculation, swirl etc.), are based on solving the transport equations for each stress component of the following form:

$$\overbrace{U_k \frac{\partial \overline{u_i u_j}}{\partial x_k}}^{\text{convection}} = \overbrace{-\left(\overline{u_i u_k}\frac{\partial U_j}{\partial x_k} + \overline{u_j u_k}\frac{\partial U_i}{\partial x_k}\right)}^{\text{production}} \overbrace{- \frac{\partial}{\partial x_k}\left[\overline{u_i u_j u_k} + \frac{1}{\rho}\left(\overline{p' u_i}\delta_{jk} + \overline{p' u_j}\delta_{ik}\right) - \nu \frac{\partial \overline{u_i u_j}}{\partial x_k}\right]}^{\text{diffusion}}$$

$$- \underbrace{2\nu\left(\frac{\partial u_i}{\partial x_k}\frac{\partial u_j}{\partial x_k}\right)}_{\text{dissipation}} + \underbrace{\overline{\frac{p'}{\rho}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)}}_{\text{redistribution}} \tag{3}$$

The above Reynolds-stress equations are not closed as the diffusion (apart from viscous diffusion), dissipation and redistribution terms need to be modelled. In the present study, the model of Speziale et al. [2] was adopted.

Chemical reactions considered here involve a simultaneous reaction of several species for which the concentration is governed by separate differential equations. The conservation equation for the mass fraction (m) of the species j can be written as:

$$\frac{\partial(\rho\, U_i\, m_j)}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\frac{\mu_t}{Sc_t}\frac{\partial m_j}{\partial x_i}\right) + S_j \tag{4}$$

where Sc_t is an effective Schmidt number and S_j is the mass rate of creation or depletion of species j.

In order to determine the temperature field within the reactor we solve the transport equation for the mixture entalpy (h) of the following form:

$$\frac{\partial(\rho\, U_i\, h)}{\partial x_i} = \frac{\partial}{\partial x_i}\left[\frac{k + k_t}{C_p}\left(\frac{\partial h}{\partial x_i} - \sum_j h_j \frac{\partial m_j}{\partial x_i}\right)\right] - \frac{\partial}{\partial x_i}\sum_j h_j J_{ji} - \tau_{ij}\frac{\partial U_i}{\partial x_j} + S_h \tag{5}$$

where k and k_t are molecular and effective conductivity, C_p is mixture specific heat capacity, J_{ji} corresponds to the flux of species j along direction i, τ_{ij} stands for deviatoric stress tensor and finally S_h is an enthalpy source term for chemical reactions.

Formation of solid particles is controlled by three separate processes; namely nucleation, surface growth and coagulation resulting in continuous particle size distribution at the reactor outlet. In order to numerically simulate this distribution, the continuous particle size distribution is discretized by a finite number of class sizes (minimum 12). Each class size is then treated as a separate chemical species whose concentration is governed by the differential equation of the same form as equation (4).

3. PARALLEL IMPLEMENTATION

The differential transport equations are transferred into their algebraic form using a standard finite volume technique. Co-located grid arrangement, which assumes the placement of all dependent variables at the centre of control volume, has been used.

The parallel implementation of the code has been developed using the Communicating Sequential Process (CSP) programming model. Each process runs its own code and communicates and synchronizes its actions with other processes by exchanging messages. Typically a single process is assigned to a single processor. The code can use a various type of message passing harnesses including Intel Specific (NX/2) for Intel i860 hypercube and PVM or MPI for IBM SP2 system.

The code is parallelised using a traditional domain decomposition strategy whereby the global domain is partitioned into rectangular subdomains which are allocated to unique processes. To ease the implementation of the computations a buffer of two grid points (Halo cells) is added to the rectangular domains to accommodate the finite difference and finite volume computational stencils on both real and subdomain boundaries. Figure 2 shows a general partitioning and how the buffer data is replicated around each subdomain. For the global data circulation (required for the calculation of residuals) the processors are assumed to map into a ring topology, each processor in the ring having an identifying index (Idnode), and data is circulated to processors with increasing index. The processor with the largest index circulates the data to the first processor.

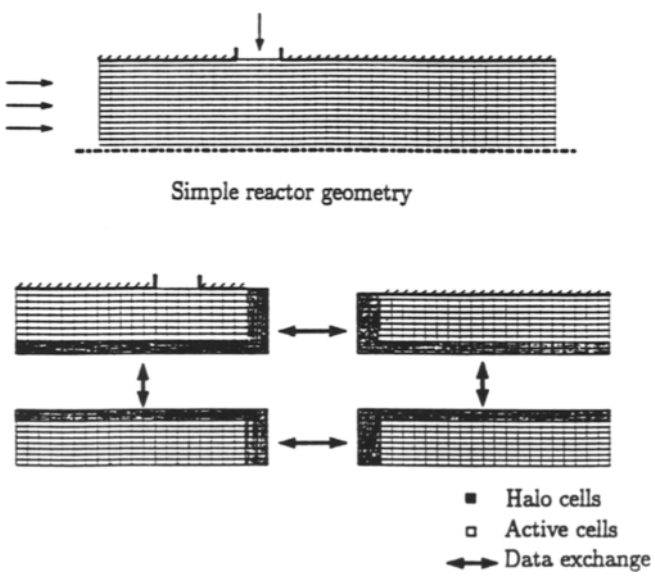


Figure 2. Allocation of grid to four subdomains

The algebraic equations generated by the process of discretization are solved using a standard Tri-Diagonal Matrix Algorithm (TDMA) for all equations apart from the

pressure correction equation. For this equation the Pre-conditioned Conjugate Gradient (PCCG) Method was used. Both Solvers are well-established within the CFD community but it is still important to emphasize a couple of points concerning their parallel implementation. The parallel implementation of the TDMA solver involves limiting the range of the back-substitution step. After each sweep of the solver the halo regions are updated and it is these field values in neighbouring subdomains which are used as boundary conditions for the back-substitution. For the parallel implementation of the conjugate gradient method it is important to mention that the communication costs are dominated by global sums as the main steps in this algorithm involve vector products.

4. PARALLEL PERFORMANCE

The target here is to calculate the flow with chemical reactions, described in Section 1.1, on various type of parallel machines. This would include Intel i860 hypercube and IBM SP2. The main factor in determining the parallel performance of the code will be an efficiency coefficient defined as: $\eta = T_1/(nT_n)$ where T_1, T_n are the execution times using one and n processors respectively. The calculation of this parameter requires first obtaining the results for the single node and bearing in mind the large number of equations (23 in conjunction with $k-\epsilon$ model or 26 when the RSM is used) which need to be solved for the present problem, a size of computational grid is determined by memory of each node. Since the Intel i860 hypercube has only 16 MBytes of memory per node it is only possible to adopt 64x16 grid size for a single processor run.

Table 1 details the performance of the code using both k-ϵ and RSM models of turbulence for different processor configurations. A previous study [3] showed that for this type of geometry (very long thin pipes) the best performances are achieved when partitioning is done along the axial direction and therefore the same approach is adopted here. The timings given are in seconds per 100 iterations for an Intel i860 hypercube (64 nodes with 16 MBytes of memory per node). The total time (Ttotal) is the sum of the communication (Tcom) times and calculation times (Tcalc). The communication times are comprised of the times required to exchange informations between processors (Thalo) and of global communication times (Tcirc).

In general both models produced a very good efficiency for the present number of processors. Results obtained by the k-ϵ model are slightly better which is probably due to the need to exchange a larger amount of data between processors when using the RSM. There is only a modest increase in total RSM computation time (compared with k-ϵ results) indicating that the present computing time is predominantly spent on calculations within the chemical model.

Table 2 gives results for the same runs but now on IBM-SP2 machine (14 thin nodes with 64 Mbytes and 2 thick nodes with 128 Mbytes of memory). The MPI message passing harness is used here for communication between processors. Results presented here are very similar in terms of efficiency compared with previous table indicating a very good

scalability of the present code.

Table 1
Total efficiency on Intel i860a

	k-ϵ model				RSM model			
	$T_{total}(s)$	$T_{calc}(s)$	$T_{com}(s)$	$\eta(\%)$	$T_{total}(s)$	$T_{calc}(s)$	$T_{com}(s)$	$\eta(\%)$
1x1	701	701	-	100	745	745	-	100
2x1	354.6	340.6	14	98.8	380.6	365.4	15.2	97.8
4x1	200	168	32	87.6	216	175	41	86.2
8x1	109	84	25	80.4	117	88	29	79.6

Table 2
Total efficiency on IBM SP2

	k-ϵ model				RSM model			
	$T_{total}(s)$	$T_{calc}(s)$	$T_{com}(s)$	$\eta(\%)$	$T_{total}(s)$	$T_{calc}(s)$	$T_{com}(s)$	$\eta(\%)$
1x1	210.2	210.2	-	100	218	218	-	100
2x1	107.2	100.5	6.7	98	113	108.6	4.4	96.4
4x1	59.5	52.6	6.9	88.3	62	54.2	7.8	87.9
8x1	33.1	27.4	5.7	79.4	34.7	28.2	6.5	78.5

5. RESULTS

This section is concerned with the results obtained for chemically reacting flows in pipe type geometries (TIOXIDE reactor). The particular reaction of interest is described in Section 1.1. Figures 3 and 4 show the contours for both reactants ($TiCl_4$ and O_2) and products (TiO_2 and Cl_2) of the main chemical reaction occurring within the reactor. The results are presented for both turbulence models. It is clear that two models lead to a different level of mixing and, therefore, the distribution of reactants and creation of the products are occurring at different positions within the reactor. For example the contours of the main product Cl_2 are more distorted along the line where the strongest mixing between the incoming horizontal jet and vertical top wall jets occur.

This difference of product and reactant concentrations within the reactor will ultimately affect the wall deposition rate as well as the size characteristics of the titania particles at the outlet. These two parameters are of critical importance since, the deposition rate would have direct effect in determining whether or not a catastrophic build-up on the wall will arise and furthermore, for any particulate product, their properties depend on the size of the individual particles created.

Temperature and $TiCl_4$ concentration profiles along reactor wall are plotted in Figures 5 and 6 respectively. Clearly, two models produce a different results so that in the case of maximum temperature point, two models differ from each other by a 100 K. Since a

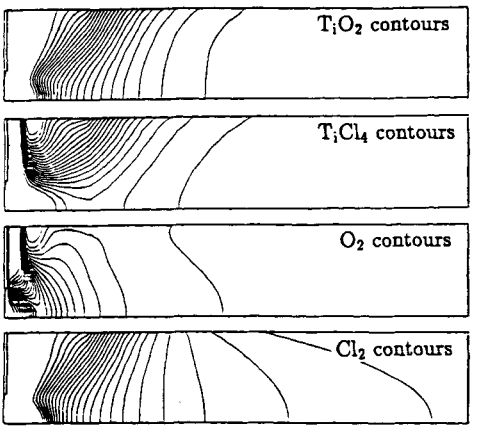


Figure 3. Chemical reactor. k-ϵ results.

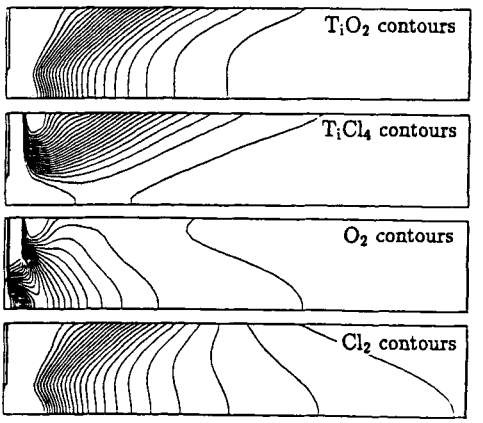


Figure 4. Chemical reactor. RSM results.

maximum metal temperature is a crucial design parameter, an error here can have serious impact on safety and costs.

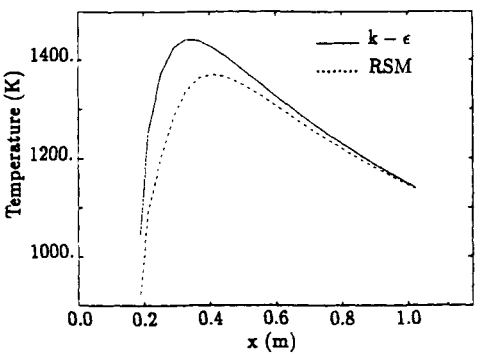


Figure 5. Temperature distribution along the wall.

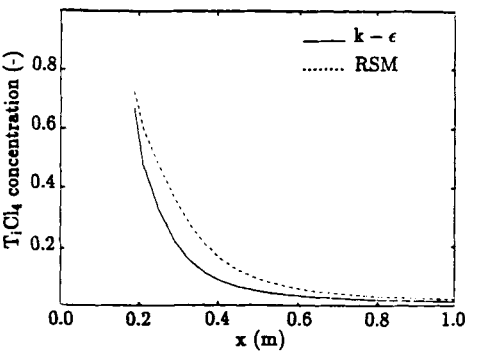


Figure 6. $TiCl_4$ distribution along the wall.

Two previously shown curves can be used to estimate a deposition rate on the wall using simple Arrhenius formula: deposition rate $\sim$ [$TiCl_4$] exp(- 20000/T). The deposition rate calculated in this way is shown in Figure 7. Two models produced a factor of 2 in the peak deposition rate which is clearly very significant.

Creation of solid particles is directly controlled by the reaction rate of each species in each particular reaction. In the present model reaction rate is chosen as minimum between

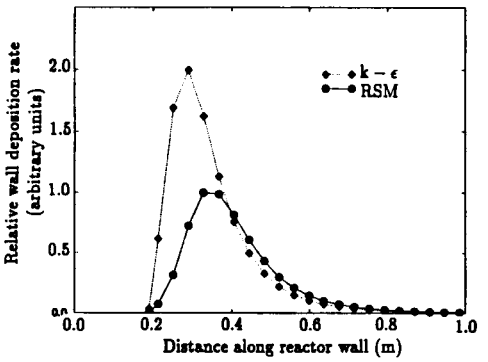


Figure 7. Wall deposition rate

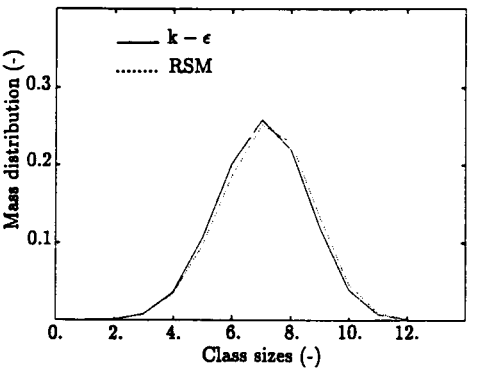


Figure 8. Mass distribution of solid particle.

standard Arrhenius reaction rate and Magnussen and Hjertager model (MH, [4]). In laminar flows the reaction rate is controlled by Arrhenius model while MH model takes into account the influence of turbulence. The MH model contains an empirical constant (C) which is assumed to be C=1 in present study. Figure 8 includes results for mass distribution obtained by both turbulence models. Clearly, RSM model alters overall mass distribution of particles compared to the k-ϵ model.

6. CONCLUSIONS

The Reynolds-stress model of turbulence has been used to calculate the chemically reacting flow. A code has been ported on two different parallel platforms (Intel i860 and IBM-SP2) and satisfactory efficiency has been achieved. It was shown through comparison with simple k-ϵ model of turbulence that a difference in turbulence field can have an important impact on prediction of some important reactor parameters. Thus, two different approaches to turbulence modelling resulted in the difference for the peak deposition rate by a factor of two and furthermore in different particle size distributions at the reactor outlet.

REFERENCES

1. Hounslow, M.J., Ryall, R.L. and Marshall, V.R., (1988), AIChe J., Vol. 34, No. 11, pp. 1821–1832.
2. Speziale, C. G., Sarkar, S. and Gatski, T. B., (1991), J. Fluid Mech. 227, p. 245.
3. Carter, J.G., Cokljat, D. and Blake, R.J., (1995), Lecture Notes in Computer Science 919, Springen-Verlag, pp. 434-440.
4. Magnussen, B.F., and Hjertager, B.H., (1976), 16th Symp. on Combustion, Cambridge, MA, Aug. 15-20, pp. 719-729.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Development of a Parallel Spectral Element Code Using SPMD Constructs

H. L. Bergman, J. S. Vetter*, K. Schwan*, and D. N. Ku

GW Woodruff School of Mechanical Engineering and *College of Computing, Atlanta, GA 30332

ABSTRACT

While computational fluid dynamics are an alternative to experimentation, the CPU time requirements for three dimensional, unsteady problems are prohibitive. The spectral element method has high spatial accuracy and converges quickly. Furthermore, the spectral element method can be parallelized without sacrificing numerical efficiency. The natural domain decomposition of the spectral element method is ideally suited for single-program, multiple-data execution.

A spectral element code was parallelized on the KSR-2 64-processor supercomputer using single-program, multiple-data constructs. Very little of the sequential code had to be modified for shared memory execution because of the natural domain decomposition inherent in the spectral element method.

The parallelized code was tested on two unsteady flow problems consisting of an axisymmetric sudden expansion at Reynolds number of 175 and a 3-D curved bifurcation model of the coronary arteries at a Reynolds number of 220. The parallel code produced a speedup of over 7 using 24 processors for the 2-D simulation. A speedup of over 3 was realized using 22 processors to simulate flow through the 3-D bifurcation.

1. INTRODUCTION

There is a strong correlation between fluid dynamics and atherosclerosis. In the presence of low, oscillatory wall shear stress, arteries are likely to occlude, causing heart attack or stroke [1]. For this reason, it is important to quantify the wall shear stress as a function of time in unsteady branched flows. Numerical studies, of bifurcated, pulsatile, 3-D flow in the coronary arteries consumed up to six weeks of computational time per simulation on a fast IBM RISC/6000 workstation [2]. Using a vector supercomputers offered little benefit. Although a Cray was a few times faster than the high-performance workstation when executing non-optimized code, jobs often waited days or weeks to reach the front of the queue. The computational fluid dynamics (CFD) code was therefore parallelized to bypass the severe time constraints of uniprocessor machines.

The spectral element method (SEM) was chosen as the basis for our numerical simulations because it has high order accuracy and can be parallelized efficiently. The SEM is a high order finite element formulation which has a natural domain decomposition. Parallelism is typically exploited by decomposition along element boundaries and incurs no convergence penalties [3-7].

This paper details the parallelization of a SEM code onto the KSR2 supercomputer. The KSR2 was selected as the target platform because it has relatively low overhead, it is easy to program, and it has the time resources required for this project. Coarse grained parallelism was implemented in a manner similar to previous work [3]. The major exception in this case was the use of a shared memory programming model. Decomposition in this style lends itself to portability onto other shared memory architectures which support single-program, multiple-data (SPMD) constructs.

2. METHODOLOGY

In the SEM, the physical domain is decomposed into several high order macroelements. In each macroelement, velocity and pressure are expanded in terms of Lagrangian interpolants at the N^2 Gauss-Lobatto-Legendre and $(N-2)^2$ Gauss-Lobatto points, respectively [7]. This creates a locally structured problem. Consequently, there is a high ratio of intraelement work to interelement communication. Typically seventh order elements are used in a coarse mesh and tenth to twelfth are used in a fine mesh.

In this specific implementation, time is advanced with a third order Adams-Bashforth scheme. The solver uses the Uzawa algorithm to separate the calculation of the pressure and velocity terms. A diagonally preconditioned conjugate gradient solver operates on the decomposed equations.

2.1. Target platform: the KSR-2

The KSR-2 is a 64 processor MIMD machine arranged in two rings of 32 cells each. Remote memory access latency on the KSR is roughly 1/20th that of the iPSC/860 [9]. Physically, the KSR is a distributed memory machine, but on a programming level the machine acts like a shared memory computer.

Because of its hierarchical structure, memory latency depends highly on the distance between the processor and the memory location [10]. For this reason it is important to keep memory references local and, when possible, keep the parallel execution running on a single ring. All the necessary data will be stored on the subcache during large parallel execution of typical CFD problems.

2.2. Profiling

Profiling the sequential code was performed prior to planning the parallelization strategy. Both a call graph and a list of where time was spent were determined by the *gprof* profiler. Gprof showed, as was expected, that most of the execution time was spent performing low-level matrix-matrix multiplication and vector addition. The call graph showed interesting results. Typically, over 90% of the execution time was spent in several high-level routines which discretize the Navier-Stokes equations.

Fortunately, these large blocks of code are performed independently on each element, usually inside a do loop spanning the space of elements. Such routines can be effectively parallelized. The discretization and conjugate gradient modules are of this type.

Other functions, such as those to calculate norms, can be parallelized but require considerable communication. While some execution time would be saved by using reduction algorithms, doing so would have consumed a disproportionate amount of human programmer time and yield only a small increase in speedup.

Routines were parallelized in descending order of run time. Those procedures (and their descendants) which consumed the most amount of run time *and* could be productively parallelized were parallelized first.

2.3. SPMD Implementation

The results of the profiling showed that there were large blocks of code that are performed in do loops spanning all K elements. The SPMD paradigm is ideal for parallel SEM because there is concurrent execution operating on different portions of the same arrays. KSR Fortran contains SPMD constructs, called *parallel regions*, which work well with the domain decomposition strategy.

Note that all of the physical parameters—collocation point location, each component of velocity, pressure, and local viscosity—are four dimensional arrays. Each thread was to be assigned a fraction of the elements to work on, consistent with the general SEM parallelization strategy. Macroelements were equally divided among threads to ensure load balancing. A do loop that was indexed in the sequential version like

```
do 10 iel=1,number_of_elements
```

would become

```
do 10 iel=istart(thread),istart(threadid)+ilength(threadid)-1
```

in the parallel version. In this manner, accidental overwriting was prevented.

The next challenge was passing variables into low level subroutines. The solution was to have each thread pass the location of the beginning of its portion of the arrays to low level subroutines. This location was specified in the call by explicitly passing a pointer to the correct register:

```
v(1, 1, 1, iel) .
```

For example, a typical block of code performing the same instruction on each element, which originally (sequentially) looked like:

```
call A(v, temp)
call B(v, temp)
```

was changed to:

```
call A(v(1,1,1,iel), temp(1,1,1,iel))
call B(v(1,1,1,iel), temp(1,1,1,iel))
```

Writing the parallel region in this manner created private copies of each thread's data on cache. These variables were thus stored locally in each

processor's memory. On the KSR, references to local registers are no less than three times faster than those to memory on other cells [9]. Keeping most references local was one important part of the parallelization strategy, which is important considering the latency characteristics of the KSR.

Common arrays were frequently used for storing temporary data in the sequential version. These arrays were originally declared as common variables to increase speed. If not managed correctly, these common arrays would be considered common among all threads on the KSR. This means that if one thread stored data in a common temporary array, it would write to the single, global copy of that array, and so would the other threads. When a thread read from this array, it would acquire the last value written to that space, which may have come from another thread. To eliminate confusion, the temporary arrays were indexed with a fourth dimension to provide each element with its own space.

2.4 Compartmentalizing the parallel constructs

The machine specific parallel directives and parallel subroutines were separated from the rest of the source code with precompilier directives. A logical variable, set at compilation time, instructed the precompiler whether to include the parallel code or not. Debugging was simplified by being able to set individual portions of a parallel code to run sequentially. Using these directives, the parallelized code can be ported to any sequential machine and run without making any changes. Efforts to port the parallel code onto other shared memory architectures would require using different specific constructs to handle the threads.

3. BENCHMARK CASES

Two sample geometries, shown in Figures 1, were created to test the code under a variety of problems. One problem was a 48 element sudden expansion with expansion ratio of 1:2. This geometry was useful because the grid is axisymmetric and orthogonal, which involved fewer computations than a curved geometry. The inlet conditions were parabolic flow at Reynolds number of 175; the exit length was 60 diameters. Figure 1(a) shows the computational grid with streamlines superimposed.

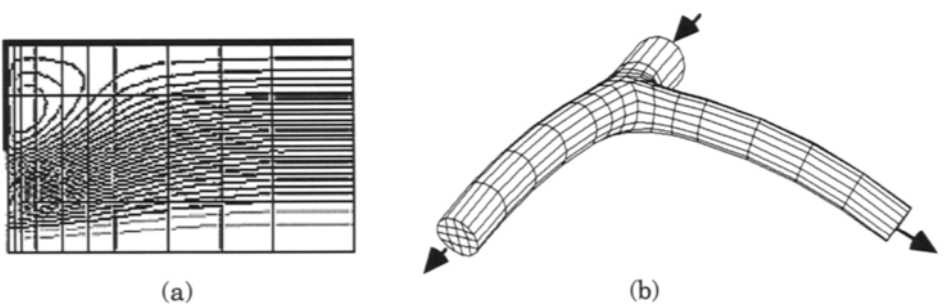


Figure 1. (a) Sudden expansion geometry with streamlines superimposed, and (b) coronary bifurcation geometry.

To see how 3-D problems were handled by the parallel solver, a complex, curved 3-D geometry was also used. This case was of more practical interest because large, 3D problems drive the need for parallel CFD. The geometry used was the 308 element coronary bifurcation designed by He [2], shown in Figure 1(b). The problem was decomposed into as many as 22 threads.

4. RESULTS

The timing results presented below were generated from the average of three runs per specific geometry and problem size. The standard deviation between runs was on the order of one second for all runs with less than 12 processors; that when using more processors was roughly 5%. The outliers were caused by high machine overhead from multiple users' large jobs.

Figure 2(a) shows the speedup results for the 48 element sudden expansion model with polynomial order of 7, 10 and 12. Note that the number of collocation points in each macroelement equals the square of the polynomial order. Thus compared to the coarse grid, the intermediate grid had roughly double the number of collocation points, a measure of computational work, and 50% more boundary nodes, a measure of communication. The finest mesh (twelfth order polynomials) had three times as many collocation points as did the coarse mesh.

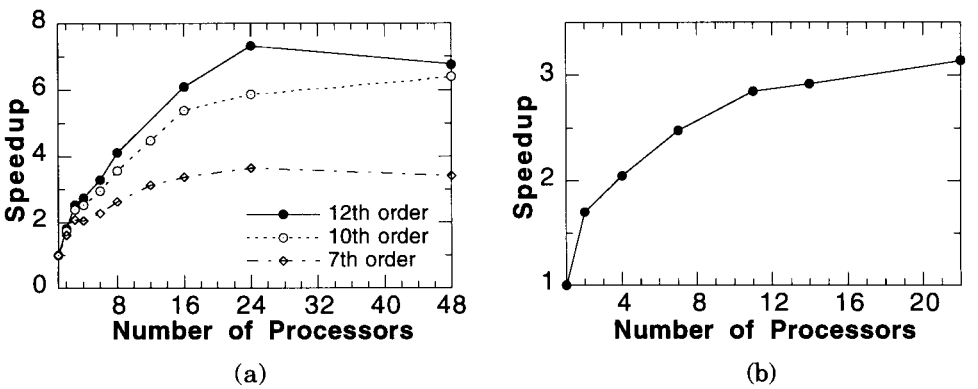


Figure 2. Speedup curves for the (a) axisymmetric sudden expansion and (b) 3D coronary bifurcation model.

The results showed that despite the doubling of computational work from seventh to tenth order elements, the speedup increased by only 50% when higher order elements were used. The improved speedup for the tenth order elements was limited by the additional communication incurred by having more boundary nodes and higher startup costs related to initially copying velocity and geometry data to threads in the parallel region. The maximum speedup observed with the twelfth order mesh was 7.3, more than double that of the seventh order mesh.

The 3-D simulation represents the maximum amount of work encountered in a CFD problem of interest to biomedical applications. Figure 2(b) shows the speedup curve for the 3-D coronary simulation. The maximum speedup of 3.2 was lower than the speedup of 3.7 obtained by the small sudden expansion model of similar element order. Speedup was limited by the large amount of data copying and communicating, despite there being many times more computations being performed. Communications were high because the preprocessor did not assign neighboring elements to the same thread. Altering the preprocessor's element numbering scheme manually is very time-intensive and thus was not performed.

5. DISCUSSION

Currently, some portions of the code are still running sequentially. The serial fraction is 18% based on the profiling data for the coarse grid sudden expansion simulation. Having serial execution, of course, limits speedup and makes performance evaluation difficult; speedup of the entire execution does not reflect the efficiency of the parallel implementation. Therefore, the speedup and efficiency *of the parallel portions of code* were measured to evaluate parallel performance. These data were produced by placing timing statements around the parallel regions of code.

Figures 3 (a) and (b) show the speedup and efficiency of the parallel portions of code when 10% and 82%, respectively, of the code was running in parallel. The efficiency of the parallel code increased dramatically as the amount of work done in parallel increased. When a small fraction of code was running in parallel, efficiency was less than 20% with a large number of processors . Presently, with over 80% of the code parallelized, the parallel portion is over 70% efficient for this coarse grid simulation when a large number of processors were used.

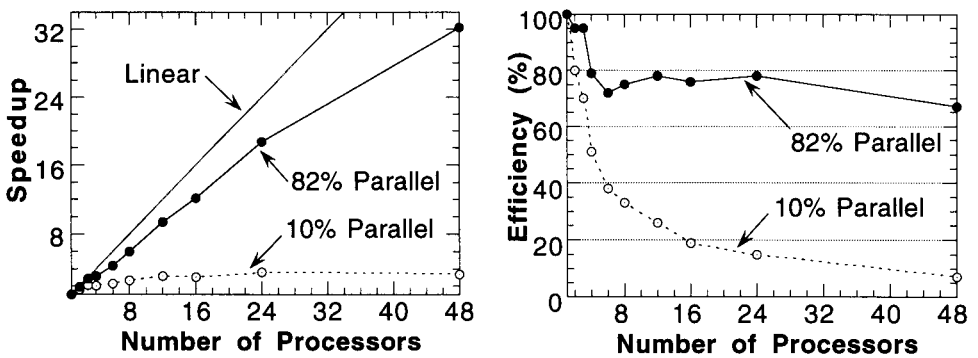


Figure 3. Speedup curves (a) and resulting parallel efficiency (b) for portions of code running in parallel. Shown are data from two coarse grid 2-D simulations: with 10% of code parallel and with 82% of code in parallel.

The parallelization process has not yet been completed. Figures 3 showed that parallel efficiency increased dramatically as the amount of parallelism increased. Therefore, as more of the code is parallelized, parallel efficiencies should be greater than 70%.

The devices and implementations required to parallelize this SEM code for the KSR are compartmentalized. There are two major benefits to this style. First, the parallelism is completely transparent when no parallelism is needed. For example, the parallel code can be ported to any other architecture and run sequentially. Second, the KSR specific constructs are easily identifiable and replaceable. This would make porting simplified because the KSR specific commands can easily be replaced by the new machine's commands.

This style of data management also simplifies porting to other shared memory machines because arrays, such as that for velocity, were treated as if there were only one copy. Parallelism was introduced by having different threads work on separate portions of the loops which span the set of elements. Parallelism on other shared memory machines can also occur at the element-loop level.

The 2-D speedup results presented here are similar to other published SEM results for comparably-sized problems [4], despite there being sequential code remaining. Even better results can be expected this shared memory SEM paradigm because about half of the remaining sequential time in the parallel code can be parallelized. Furthermore, parallelizing these smaller functions will increase parallel efficiency by reducing the number of barriers needed in the currently-incomplete parallel region. The speedup results show that the SPMD programming style is well-suited to domain decomposition parallelism.

6. CONCLUSIONS

Parallel regions, or the generic "single-program, multiple-data" constructs were useful for porting a spectral element code to a shared memory model. The speedup results were comparable to those from porting a similar code onto a hypercube, despite there being some sequential code remaining.

As more of the code becomes parallelized, the maximum speedup curve will shift upwards. It was shown that as more parts of the code were parallelized, the efficiency increased. Therefore, it is expected that a considerable increase in speedup could occur when more parallelism is introduced into the code.

Modifications to the code were minimized by using a shared memory programming approach. Only small changes should be necessary to port the code onto other shared memory parallel architectures.

REFERENCES

1. Ku DN, et al (1985) Pulsatile flow and atherosclerosis in the human carotid bifurcation: Positive correlation between plaque location and low and oscillating shear stress. *Arteriosclerosis* 5, 293-302.

2. He X and Ku DN (1994) Flow in the human left coronary artery: effects of variations in bifurcation angle. *Advances in Bioengineering* BED - Vol. 28, 435-436.
3. Fisher, PF (1990) Analysis and application of a parallel spectral element method for the solution of the navier-stokes equations. *Computer Methods in Applied Mechanics and Engineering* 80, 483-491.
4. Fisher, PF, et al (1988) Recent advances in parallel spectral element simulation of unsteady incompressible flows. *Computers & Structures* 30, 217-231.
5. Ku HC, et al (1989) A pseudospectral matrix element method for solutioin of three-dimensional incompressible flows and its parallel implementation. *J. of Computational Physics* 83, 260-291.
6. Tan CS (1989) A multi-domain spectral computation of three-dimensional laminal horseshoe vortex flow using incompressible Navier-Stokes Equations. *J. of Computational Physics* 85, 130-158.
7. Patera AT (1984) A spectral element method for fluid dynamics: laminar flow in a channel expansion. *J. of Computational Physics* 54, 468-488.
8. Fisher, PF and Patera, AT (1991) Parallel spectral element solution of the Stokes problem. *J. of Computational Physics* 92, 380-421.
9. Dunigan TH (1992) Kendall Square Multiprocessor: early experiences and performance. US Department of Energy, Engineering Physics and Mathematics Division, Report ORNL/TM-12065.
10. Kendall Square Research (1991) *KSR1 Principles of Operations.* Kendall Square Research, Waltham, MA.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel Computation of Turbulent Reactive Flows in Utility Boilers

J. Lepper

Institute for Process Engineering and Power Plant Technology, University of Stuttgart †,
Pfaffenwaldring 23, D-70550 Stuttgart, Germany.
Email: lepper@ivd.uni-stuttgart.de

The paper presents the parallel implementation of a simulation program for turbulent reactive flows in utility boilers on an Intel Paragon parallel computer. The explicit parallelization with the SPMD programming model and synchronous message passing requires only few modifications of the existing code. The static decomposition of the structured grid topology is done for an isothermal boiler model and a non-isothermal full-scale boiler configuration. For both cases a domain decomposition method with two different approaches is used to subdivide the numerical grid. If geometrical aspects are the basis for the subdivisions, only moderate speedups with limited numbers of subdomains are achieved, whereas decompositions regarding a better load balance result in higher speedup. With an additional data exchange during the implicit solution of the pressure correction equation, all computed results show a convincing numerical behavior.

1. INTRODUCTION

The numerical prediction of phenomena within utility boilers requires submodels for fluid flow motion, turbulence, all combustion processes and the radiative heat transfer. All of these models are strongly coupled and interact in complex ways. Due to the dimensions of utility boilers and the time and length scales of the important phenomena an enormous amount of memory and computational power is necessary to obtain numerical predictions with a high level of accuracy in reasonable computing times during the design phase.

To meet the first requirement fine discretizations are essential, which can hardly be achieved if structured Cartesian grids are used. One remedy is the usage of a domain decomposition method for sequential applications to allow locally refined meshes in the near-burner region [1],[2]. But even with this procedure grid-independent solutions are scarcely possible due to memory limitations. Additionally the simulation times increase significantly with finer grids, and more efficient numerical algorithms have to be considered.

MIMD parallel computers with distributed memory offer more memory in a scalable way and faster turn-around times for appropriate applications. The aim is therefore to benefit from these advantages with the parallelization of an already existing application without the need to re-implement most parts of the code [3]. Most engineering programs have been developed over

† This research work is supported by the Deutsche Forschungsgemeinschaft (DFG) with a grant in the frame of a "Graduiertenkolleg" at the University of Stuttgart.

a long period of time and a re-implementation of these large codes is generally not practical.

With some kind of a domain decomposition technique coarse grain parallelism is introduced and a transformation into a parallel program becomes feasible. In this approach, also known as "data distribution" [4], the number of processors and the load balance determine the number of subdomains, whereas the geometry is the most important aspect for the other approach of the domain decomposition method. For the design of utility boilers with complex swirl burners, decompositions taking geometrical aspects into account are often favorable.

Besides the speedup, the scalability is one of the most important aspects for a parallel application [5,6]. Especially for the simulation of utility boilers, a scalable parallel application will allow finer numerical grids that are otherwise not possible due to memory limitations. On the other hand, an acceptable speedup is necessary to make use of the scalability.

The following two sections give a brief review of the numerical solution method in the sequential code and the parallelization strategy. After that computational results for an isothermal boiler model and a non-isothermal full-scale boiler are presented. All calculations have been carried out on an Intel Paragon using the NX and PVM message passing environment.

2. REVIEW OF THE SEQUENTIAL ALGORITHM

The three-dimensional simulation requires the solution of equations for the conservation of momentum and energy, for all species of the reaction model and for the radiative heat transfer. For the description of the fluid mechanics a time-averaged formulation of the Navier-Stokes equations is used (1). For the sake of simplicity all overbars are omitted in the formulas.

$$\frac{\partial}{\partial x_j}(\rho U_j U_i) = \frac{\partial}{\partial x_j}\left(\mu\left(\frac{\partial U_i}{\partial x_j}+\frac{\partial U_j}{\partial x_i}\right)\right) - \frac{\partial}{\partial x_j}(\rho u_i u_j) - \frac{\partial p}{\partial x_i} + \rho g_i \tag{1}$$

The standard k,ε-closure is applied to model the Reynolds stress tensor in (1). The coupling of the velocity and pressure field for these weakly compressible flows is given through a pressure correction equation according to SIMPLEC [7].

For the discretization a cell-centered finite volume formulation on a non-staggered grid arrangement is used. Care must be taken for the reconstruction of the convective fluxes at cell faces from adjacent cell values. A pressure-weighted interpolation method introduced by Rhie and Chow [8] is used as a correction term for linear interpolated convective fluxes. This method requires two cell values on each side of a cell face and therefore a special treatment at all boundaries. For this reason an additional overlap of internal cells at adjacent subdomains is provided.

The pulverized coal combustion is described with a reduced 4-step reaction scheme [9]. With a Eularian approach the nine species from this scheme can be written in the generalized form of a transport equation (2), which is applied to all other conservative variables, too.

$$\frac{\partial}{\partial x_j}(\rho U_j \varphi) = \frac{\partial}{\partial x_j}\left(\Gamma_\varphi \frac{\partial \varphi}{\partial x_j}\right) + S_j \tag{2}$$

The resulting system of algebraic equations is solved in a decoupled way. The relaxation within the inner iterations is done with the SOR method or with an incomplete lower-upper (ILU) decomposition solver proposed by Stone [10]. These iterations are not solved to a high

level of accuracy, because outer iterations for the whole system have to be performed to take the non-linearity and the strong coupling between the variables into account.

For the radiation, which is the predominant mechanism of heat transfer in utility boilers, a multiflux model [11] is applied. This model can be brought into a formulation according to (2), so that no additional difficulties occur during the parallelization.

3. PARALLELIZATION STRATEGY

The explicit parallelization with the SPMD programming model and message passing should change the existing program and data structure as little as possible. Especially the data structure should be maintained, because this structure has been optimized for vector processing. The treatment of connection boundaries at adjacent subdomains is therefore done in a similar way physical boundary conditions are handled. This implementation leaves the numerical kernel nearly unchanged, but it does not allow an implicit treatment at connection boundaries because individual cells cannot be addressed directly. All necessary data along these artificial boundaries are consequently updated at once after each inner iteration using synchronous message passing subroutines.

Of course, this treatment of boundaries changes the convergence behavior of the numerical solution algorithm in the parallel program [12]. The calculation within each subdomain remains implicit, while the data update at connection boundaries introduces explicit values. With an increasing number of subdomains these explicit values become more important for the solution process and the number of outer iterations increases.

An improved coupling of adjacent subdomains requires an additional data exchange within inner iterations. This extra communication for the pressure correction equation results in a reasonable reduction of the number of iterations [13,14,15]. For other variables no improvement through this stronger coupling has been found so far.

Because different types of parallel computers with distributed memory are available, it is important to obtain a portable parallel application [16]. For this an indirect implementation of the message passing subroutines has been proven useful to hide vendor-specific library calls.

For the static subdivisions of the numerical grid a domain decomposition method with two different approaches is used. In the first approach, the subdivisions are performed with regard to geometrical aspects as for sequential methods. This offers a high flexibility for geometrical variations during the design phase of a utility boiler. However, the number of possible subdomains for a parallel computation is limited, and hence the scalability, too.

In the second approach, the subdivisions are done in a way that should yield a better load balance. Since no geometrical considerations are taken into account, this method is more useful for parallel computations where the geometry is not the main subject of the simulation. Perfectly balanced subdomains are not always possible for a given three-dimensional discretization, especially due to the internal overlap and the ash hopper.

4. RESULTS

The parallel application is implemented on a 72-node Intel Paragon. For verification and test purposes an isothermal boiler model with a height of 2m and a basal surface of about 0.4m

squared is used. A relatively fine numerical grid with about 40,000 internal computed cells is applied for the computations to verify the results and test the numerical behavior for several decompositions.

The scale of this model is 1:20 compared with the full-scale boiler, which has a height of 28m and a cross section of about 64m^2. These non-isothermal simulations are done on a quite coarse grid with just about 81,000 internal computed cells. Hereby the computational gain from the parallel application, e.g. the reduction of the execution time, was of major interest.

4.1. Isothermal boiler model

The subdivisions are first made with regard to the geometry. Each burner with the assigned air-staging nozzles is kept within one subdomain. This results in three configurations with a maximum of 24 subdomains, whereby none of the decomposition led to a good load balance. For 24 subdomains the number of internal computed cells differs by a maximal factor of 2.2.

A second approach should yield a better load balance and subdivisions with up to 52 subdomains were obtained. Again no perfect load balance was possible. For the case with 52 subdomains the number of internal computed cells still differs by a maximal factor of 1.4.

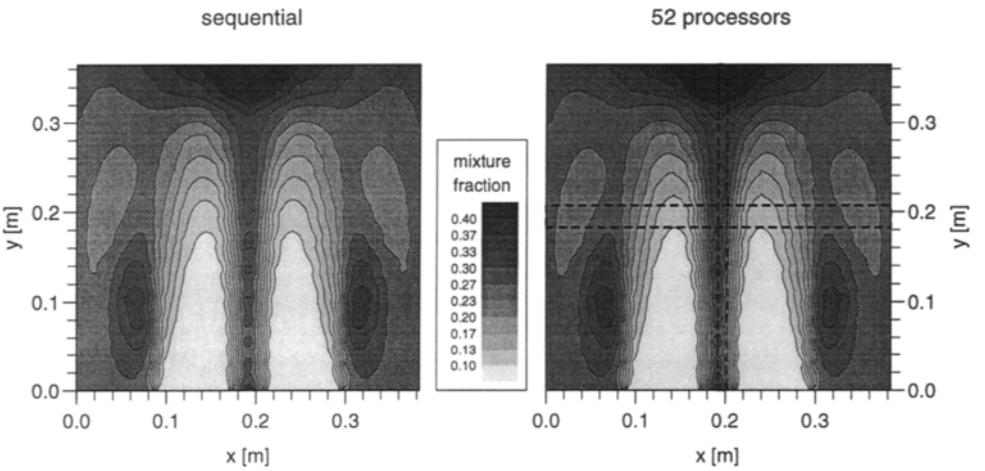


Figure 1. Mixture fraction in a cross section on the top burner level for 1 and 52 domains.

In figure 1 the mixture fraction is presented as one strongly conservative quantity in a cross section on the top burner level for a sequential computation and one with 52 subdomains. There is neither an influence visible of the internal overlap (shown as the dashed line) nor are pronounced differences obvious for the parallel computation.

Figure 2 shows the normalized mass residual versus the number of iterations as a measure for the numerical stability of the iteration process. In this figure the stability for computations with 24 and 52 subdomains is compared with the sequential run. The explicit values at the connection boundaries of the implemented data exchange for the numerical algorithm cause the oscillations for the parallel computations.

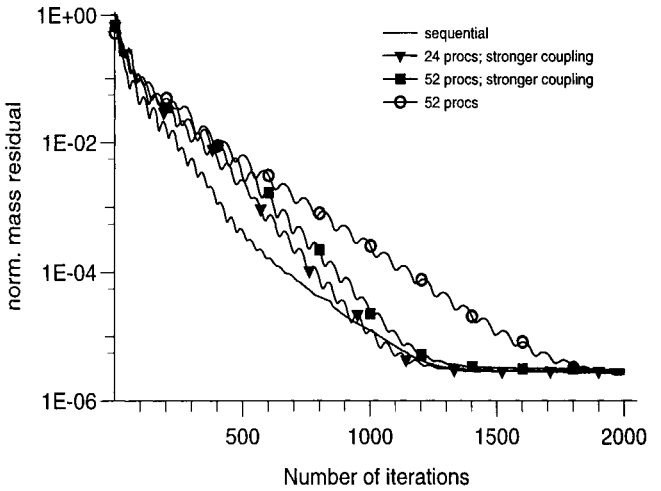


Figure 2. Numerical stability of two selected parallel computations.

With the stronger coupling within the solution of the pressure correction equation no significant increase in the number of outer iterations to reach the same converged solution is noticeable. If this coupling is omitted, the undesirable increase in the number of iterations is visible, here exemplified for 52 subdomains. Since the reduction in communication time is negligible compared with the tremendous loss in the numerical efficiency, only computations with the stronger coupling are used in the following.

The measured speedup of the calculations is shown in figure 3. The chosen discretization allowed the computation on one compute node of the Paragon to provide the reference time.

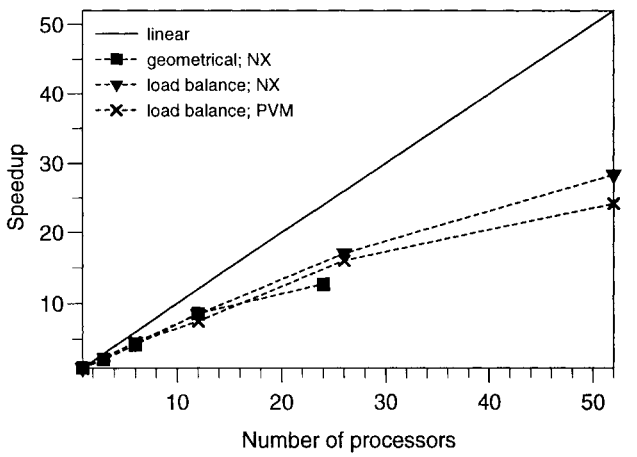


Figure 3. Speedup for the boiler model.

As expected, the results for subdivisions regarding the load balance are better than those regarding the geometry due to the disadvantageous load balance of the latter. The homogeneous computing environment on the Paragon led to only slightly less effective results with PVM.

4.2. Non-isothermal full-scale boiler

For the subdivisions, the two previously described approaches are applied again, and in figure 4 one of the geometrical decompositions with the maximum number of useful subdomains is shown.

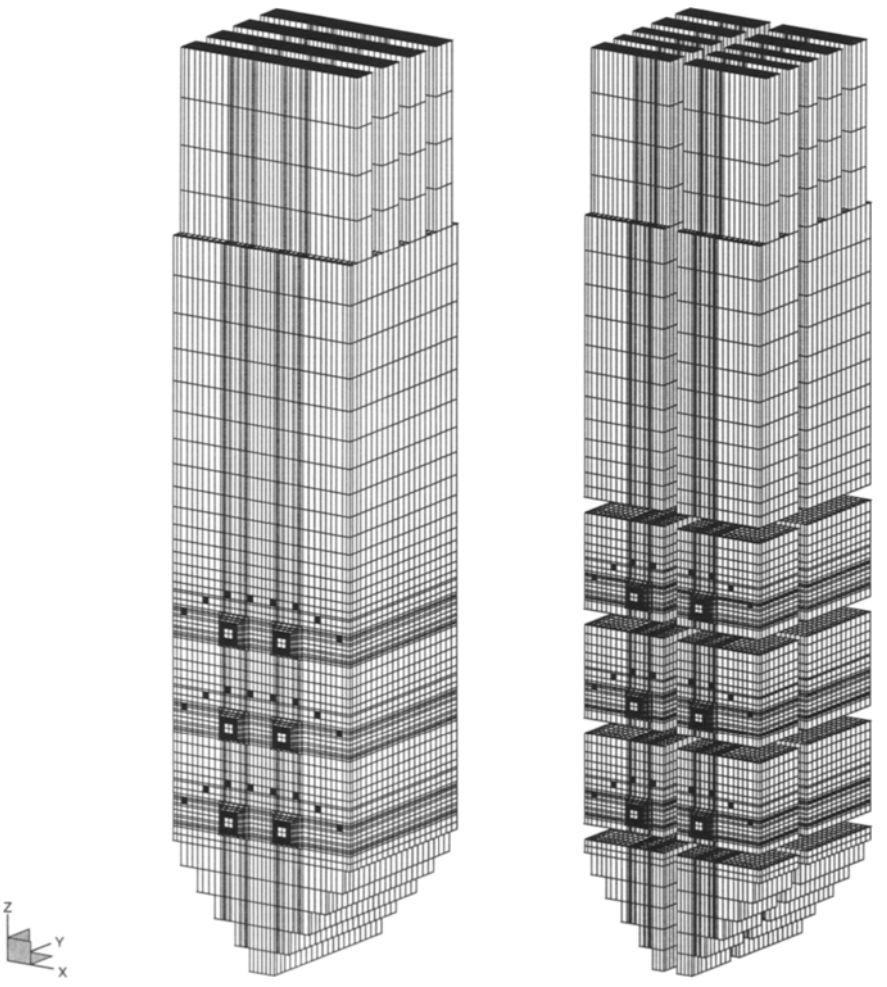


Figure 4. Numerical grid for the boiler and the geometrical subdivision into 20 subdomains.

With this procedure each of the six swirl burners with the assigned air-staging nozzles is again kept in one subdomain. It is obvious that the number of internal computed cells differs significantly, for this case by a maximum factor of about 6, which is apparently not optimal.

In the second approach subdivisions for a better load balance have been carried out up to 32 subdomains. But as in the isothermal case no optimal load balance was possible; the number of internal computed cells still differs by a factor of 1.4.

During the simulation of this boiler the burner quarl length has actually been varied, which was supported by the geometrical decomposition. Here only the subdomains containing the burners had to be modified and not the whole computational domain.

The performance of these computations using the NX message passing environment is depicted in the next figure.

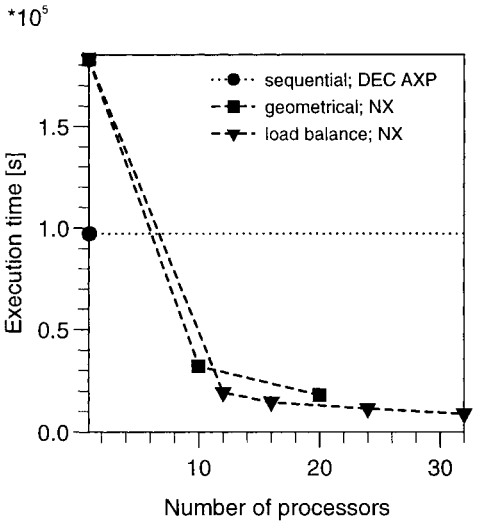


Figure 5a. Execution (wall-clock) times.

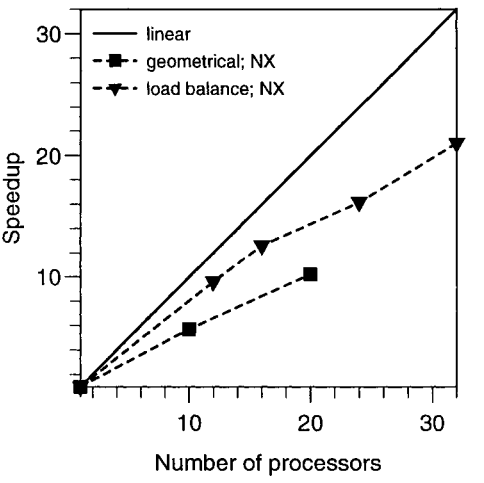


Figure 5b. Speedup (estimated).

For this configuration a computation with less than ten processors was not possible due to memory limitations on single compute nodes of the Paragon. Therefore, comparisons between a workstation (DEC 3000 Model 300X AXP) and one compute node have been done to determine the reference time on the Paragon. It appeared that the workstation is between 2.0 and 2.4 times faster than a single node of the Paragon. All values given here are based on the factor of 2.0. For comparison the execution time on the workstation is plotted, too.

The reduction of the execution times (figure 5a) with increasing number of subdomains/ processors is remarkable, even compared with the workstation. The subdivisions regarding the geometry are, as expected, less effective, a fact which becomes more distinct when looking at the speedup (figure 5b). It must be stated again that these speedups are not based on measurements for the execution time on one node of the Paragon.

The speedup is apparently not optimal, but it is acceptable for the chosen communication

pattern and the data exchange. For the user the reduction in the computing times is generally more important than the speedup, a fact which is a measure for the implementation quality of the parallel application. On 32 processors the converged solution is reached in about 3 hours and on 20 processors within 5 hours, compared to approximately 27 hours on the workstation.

However, for better parallel efficiencies more expensive changes of the program and data structure will become indispensable.

5. CONCLUSIONS

The parallel version of a simulation code for turbulent reactive flows in utility boilers has been implemented on a distributed memory parallel computer. It could be shown how an existing application can be mapped to a parallel computing environment.

The explicit parallelization of the existing application requires only some minor modifications of the program and data structure. An additional data exchange within some parts of the solution algorithm leads to the same numerical stability as for sequential calculations. The indirect implementation of the necessary message passing calls provides a portable parallel application.

For the subdivision of the numerical grid a domain decomposition method with two different approaches is used. For the first procedure geometrical aspects are the basis for the decomposition, whereas a better load balance is the basis for the second approach. Both methods show an acceptable though not optimal speedup, whereby the results of the latter procedure are better, especially for computationally expensive cases. This method should therefore be preferred if the geometrical flexibility of the first procedure is not necessary during the simulation.

REFERENCES

1. R. Schneider et al., Proc. Comp. Fluid Dynamics '94, Wiley, New York, 1994, 823.
2. P.J. Coelho and M.G. Carvalho, J. Num. Methods Eng., 36 (1993) 3401.
3. S.P. Johnson and M. Cross, Appl. Math. Modelling, 15 (1991) 394.
4. E.F. van der Velde, Concurrent Scientific Computing, Springer, New York, 1994.
5. R.K. Agarwal, Comp. Methods in Applied Sciences, Elsevier, Amsterdam, 1992, 1.
6. A.K. Stagg et al., AIAA Journal, 33 (1995) 102.
7. J.P. van Doormal and G.D. Raithby, Num. Heat Transfer, 7 (1984) 147.
8. C.M. Rhie and W.L. Chow, AIAA-82-0998 (1982).
9. W. Zinser, Fortschrittsberichte Reihe 6, Nr. 171, VDI, Düsseldorf, 1985.
10. H.L. Stone, SIAM J. Num. Anal., 5 (1968) 530.
11. A.G. De Marco and F.C. Lockwood, Proc. Italian Flame Days, (1975) 184.
12. D. Drikakis et al., J. Fluids Engineering, 116 (1994) 835.
13. M. Kurreck et al., Notes in Num. Fluid Mechanics 47, Vieweg, Braunschweig, 1993, 157.
14. M.E. Braaten, J. Num. Methods Fluids, 10 (1990) 889.
15. M. Peric et al., Proc. Parallel CFD '91, Elsevier, Amsterdam, 1992, 297.
16. A. Bode, Notes in Num. Fluid Mechanics 47, Vieweg, Braunschweig, 1993, 7.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

A Comparison of Different Levels of Approximation in Implicit Parallel Solution Algorithms for the Euler Equations on Unstructured Grids

C. W. S. Bruner[a] and R. W. Walters[b]

[a]Aerodynamics and Performance Branch, Code 432100A, Mail Stop 2, Building 1403, Naval Air Warfare Center, Aircraft Division, Patuxent River, MD 20670-5304

[b]Professor, Department of Aerospace and Ocean Engineering, 215 Randolph Hall, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061

A fully implicit algorithm for the Euler equations of fluid dynamics is implemented on a distributed-memory parallel computer. In this implementation, the computational grid is distributed across multiple processors. The Euler implicit algorithm for the solution of the governing equations gives rise to an inner matrix problem. This inner problem is solved iteratively using the symmetric block Gauss-Seidel algorithm. In the interface region between domains, one must choose between four levels of approximation to the flux Jacobians: complete communication of the evolving update across domains; communication of diagonal blocks and residual across domains outside the time integration loop; approximation of diagonal blocks and residuals using local quantities; or neglecting the non-local Jacobians and residuals completely. Also, a general method for extending one-dimensional limiters to the unstructured case is proposed.

1. INTRODUCTION

Implicit schemes for the Euler equations of fluid dynamics give rise to an inner matrix problem. When implementing these schemes on a distributed-memory architecture, compute nodes must have knowledge of the current solution and residual vector in the first non-local cell across a shared face; the converged solution is dependent only on these quantities. To enhance (or, in some case, achieve) convergence when computing on multiple compute nodes, several ways of dealing with the non-local blocks in the left-hand side matrix and residual vector may be considered. Comparison of different schemes for handling these non-local terms is the thrust of this paper.

2. FORMULATION

2.1 Description of the implicit algorithm on one processor

The computer code used for this investigation is a fully conservative upwind cell-centered finite-volume primitive-variable Euler solver; variable extrapolation (MUSCL[1]) is used for higher-order fluxes. In this investigation, Roe's approximate Riemann solver[2] is used to compute the fluxes through a face. Euler implicit time integration gives rise to a large, sparse matrix problem at every timestep:

$$\left[\frac{\Omega}{\Delta t}\boldsymbol{I}+\left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{Q}}\right)^{n}\right]\Delta^{n}\boldsymbol{Q}=-\boldsymbol{R}^{n} \tag{1}$$

Since the code solves for and stores primitive variables, $\Delta^{n}\boldsymbol{q}$ is computed explicitly from

$$\Delta^{n}\boldsymbol{q}=\frac{\partial \boldsymbol{q}}{\partial \boldsymbol{Q}}\Delta^{n}\boldsymbol{Q}$$

Second-order fluxes are computed using gradient-based reconstruction[7]; gradients for each component of the solution are computed for each cell using the discrete form of the gradient theorem[3]:

$$\overline{\nabla q_i}=\frac{1}{\Omega}\sum_{j=1}^{Nfaces}\left(q_i\hat{\boldsymbol{n}}S\right)_j \tag{2}$$

where $\left(q_i\right)_j$ is the distance-weighted average of q_i in the cells adjoining face j. These gradients are then limited so that each extrapolated solution variable is on the interval $\left[q_i^{min},q_i^{max}\right]$ for each face of the cell[4].

The flux Jacobians are computed numerically using central differences. Implicit boundary conditions are used in this investigation and are computed numerically. Only first-order spatial accuracy is used on the left-hand side; second-order accuracy is still obtained for the converged solution as long as the right-hand side of Eq. 1 is second-order accurate. The resulting sparse matrix problem (Eq. 1) is solved using Block Symmetric Gauss-Seidel (SGS)[5] iteration. The inner matrix problem can be solved to arbitrary accuracy or can be iterated a fixed number of times.

2.1.1 Limiters

All popular one-dimensional limiters may be expressed in terms of the ratio of consecutive differences[6],

$$r=\frac{u_{i+1}-u_i}{u_i-u_{i-1}} \tag{3}$$

Eq. 3 may be recast in the following form:

$$r=\frac{\left[u_i+\frac{1}{2}\left(u_{i+1}-u_i\right)\right]-u_i}{\left[u_i+\frac{1}{2}\left(u_i-u_{i-1}\right)\right]-u_i}=\frac{C}{U} \tag{4}$$

and we see that Eq. 3 may be interpreted as the ratio of central to one-sided differences evaluated at *i*.

To extend this concept to unstructured grids, we may define the denominator as the upwind or upwind-biased reconstruction to a given face minus the value in the given cell:

$$U = u_{face} - u_{cell} \tag{5}$$

Note that any one-sided reconstruction (k-exact reconstruction[7], for example) may be used to evaluate u_{face}.

Similarly, the numerator may be expressed as half the difference between the values in the two cells which share the given face:

$$C = \frac{1}{2}\left(u_{neighbor} - u_{cell}\right) \tag{6}$$

This formulation reduces to Eq. 3 for a uniform one-dimensional grid.

Eqs. 4, 5, and 6, when substituted through Eq. 3 into any of the standard one-dimensional limiter formulas, restrict the gradients in such a way that the extrapolated values at the cell face are between the values in the adjacent cells. In most unstructured formulations, it is customary to limit the extrapolated values at each face to lie between the minimum and maximum values over the cell and its neighbors; Aftosmis et al.[8] have found that limiting based solely on adjacent cell values (as in a structured grid) impedes convergence. Therefore, define

$$C = \begin{cases} \frac{1}{2}\left(u_{min} - u_{cell}\right) \text{ if } U < 0 \\ \frac{1}{2}\left(u_{max} - u_{cell}\right) \text{ if } U > 0 \end{cases} \tag{7}$$

Note that the factor ½ ensures that $r = 1$ for a linear function on a uniform one-dimensional grid.

In the cases run by the authors, the current extension to the modified van Albada limiter[9] gives slightly better convergence than the Venkatakrishnan limiter[10]; the solution is also somewhat smoother. The extended van Albada limiter is used throughout the rest of this investigation when a limiter is necessary to ensure convergence.

2.2 Description of the extension to multiple processors

Each compute node reads its own portion of the grid file at startup. Cells are divided among the active compute nodes at runtime based on cell ID; only faces associated with local cells are read. Faces on the interface surface between adjacent computational domains are duplicated in both domains. Solution variables are communicated between domains at every timestep. Fluxes through the faces on the interface surface are computed in both domains. Communication of the solution across domains is all that is required for first-order spatial accuracy, since $\boldsymbol{q}_L$ and $\boldsymbol{q}_R$ are simply the cell averages to first order. If the left and right states are computed to higher-order, then $\boldsymbol{q}_L$ and $\boldsymbol{q}_R$ are shared explicitly with all adjacent domains. The fluxes through each face are then computed in each domain to obtain the residual for each local cell.

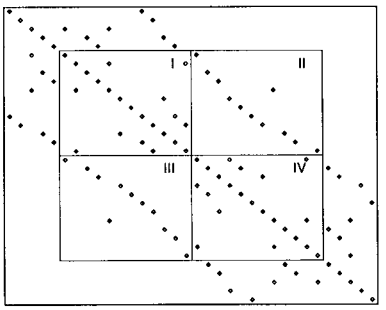


Figure 1: Closeup of domain interface region.

2.3 Cell Reordering

On the Intel Paragon, the message latency is usually the largest part of the total communication time for the medium-sized problems considered in this paper. Therefore, it is better to minimize the number of messages sent than the total number of bytes sent. Given this, a Cuthill-McKee-type[11] reordering of the grid should be close to optimal, since this ordering seeks to minimize matrix bandwidth. Because the optimal ordering (by this definition) is independent of the number of compute nodes, the reordering may be done offline.

The Gibbs-Poole-Stockmeyer variant[11] of the Cuthill-McKee algorithm was used to reorder the cells for all cases run.

2.4 Description of the domain interface region

For simplicity, consider the case of two processors, A and B. Figure 1 shows a close-up view of the domain interface region for this case. Off-diagonal blocks in zones II and III relate cells in both domains. Because A and B share solutions at every timestep, A has full knowledge of all of the blocks in zones I, II, and III, while B has full knowledge of the blocks in zones II, III, and IV. Because A and B communicate through the solution vector, it does not affect the steady-state solution for A to neglect blocks in zones II, III, and IV (similarly for B); however, one might expect the convergence to be enhanced by somehow including the effect of these blocks.

Using only the completely local blocks in the boxed-in area (zone I for A and IV for B), and neglecting all non-local blocks, is referred to as Level I approximation in this paper.

A slightly more sophisticated way of dealing with the non-local blocks is for A to again neglect the off-diagonal blocks in zones II, III, and IV, but to approximate the diagonal blocks and right-hand side of zone IV by the face-adjacent neighbor diagonal blocks from zone I and by the face-adjacent parts of A's residual vector. Then we can solve for $\Delta^n \boldsymbol{Q}$ corresponding to B's cells outside the inner iteration loop. This is simply one block-Jacobi iteration for the non-local terms in $\Delta^n \boldsymbol{Q}$. Given these terms in $\Delta^n \boldsymbol{Q}$, we can include the approximate effects of the off-diagonal blocks in zone II, which pass over to the right-hand side for relaxation schemes. B would approximate zone I quantities to estimate the effect of blocks in zone III. This is referred to as Level II approximation.

In a similar way to Level II, we can share the diagonal blocks in zones I and IV, as well as the corresponding residual vectors, by explicit communication between A and B. Then we can solve for $\Delta^n \boldsymbol{Q}$ as before. This is Level III approximation.

Finally, if A communicates to B the components of $\Delta^n \boldsymbol{Q}$ which are non-local to B at every inner iteration, and vice versa, then the parallel implementation should have convergence similar to the serial version. If practical computational concerns did not suggest reordering the local cells, or if a block Jacobi scheme were used for the inner problem, convergence would be identical. However, it might be expected that the substantial increase in communications overhead associated with this scheme may more than offset any gains. Note that this

scheme is the only scheme which includes the whole effect (through the evolving $\Delta^n \boldsymbol{Q}$) of all of the blocks in the domain interface region. This scheme is Level IV approximation.

3. DESCRIPTION OF THE TEST CASES

Two test cases were used in this investigation. The two-dimensional case was run on a grid containing wedges only. The fluxes through the faces corresponding to the out-of-plane direction were not computed. After being initialized with a converged first-order solution, each case was computed using second order fluxes. The inner problem was converged such that

$$\frac{\|\boldsymbol{Ax} - \boldsymbol{b}\|_1}{\|\boldsymbol{b}\|_1} \leq innerTol$$

or

$$\|\boldsymbol{Ax} - \boldsymbol{b}\|_1 < \text{DBL\_EPSILON}$$

or after *innerIters* inner iterations, whichever occurred first.

Neither of the test cases would fit in physical memory on one compute node of the Paragon; the performance on one compute node was calculated from the performance on a Silicon Graphics Indigo[2] and the ratio of CPU speeds between the SGI machine and one node of the Paragon. This ratio was calculated from the time to converge a 2-D supersonic ramp problem on each machine using one compute node.

Note that problem-size dependent differences in cache performance on each machine are not accounted for. Therefore, efficiencies of more than 100% are possible.

For each case and approximation level, the CFL number used was the largest that could be run on 32 compute nodes, but there was no "tweaking": the CFL number used was the largest of the sequence 1, 2, 5, 10, 20, 50, ..., that would converge. The CFL number was not changed as the number of compute nodes varied. This makes it easier to spot trends, but is somewhat sterile: in a practical environment, the CFL number would always be the largest runnable for the number of compute nodes.

3.1 RAE 2822 airfoil

This is a transonic 2D case with a multiply-connected domain. Mach 0.73 flow is computed over an RAE 2822 airfoil at 3.19° angle of attack. A shock forms on the upper surface of the airfoil for this case. Excellent experimental data for this case may be found in Cook, et al.[12]

This case was converged six orders from the converged first-order solution. The inner problem was converged to 1×10^{-7}, and was limited to 100 inner iterations. The grid has 11,848 cells and 17,873 active faces.

3.2 ONERA M6 wing

This ubiquitous three-dimensional test case involves transonic flow with coalescing shocks; see Schmitt, et al.[13] for details. This case is run at Mach 0.84 and 3.06° angle of attack.

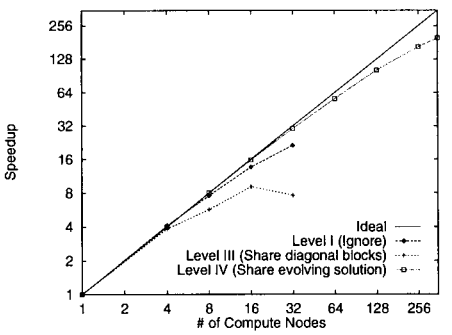


Figure 2: Speedup for the RAE 2822 airfoil case. The data point at the far right is for 352 nodes.

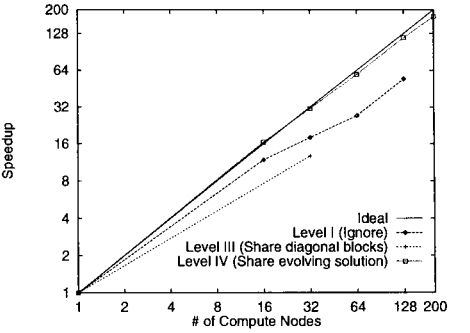


Figure 3: Speedup for the ONERA M6 wing case.

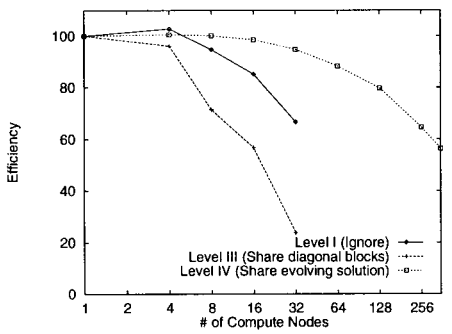


Figure 4: Efficiency for the RAE 2822 case.

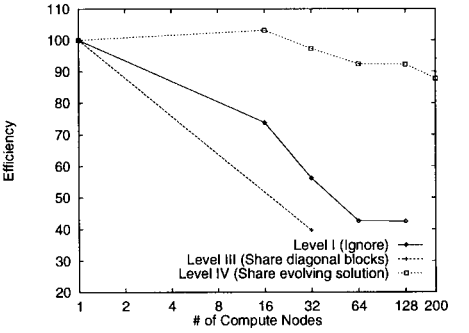


Figure 5: Efficiency for the ONERA M6 wing case.

Due to CPU-time constraints, the timing runs for this case were converged only two orders of magnitude from the converged first-order solution. Also, the inner problem was only converged to 1×10^{-5} and was limited to a maximum of 50 iterations. The grid has 96,207 cells and 196,652 faces.

4. RESULTS AND DISCUSSION

The performance of the Level II approximation was so poor compared to the other methods that results for this scheme are not presented here. The reason for the poor performance was the very low CFL number required for convergence.

Figure 2 and Figure 3 show the relative speedup of the various schemes for each test case. The RAE 2822 case needed no limiting; the extended van Albada limiter described above was used for the ONERA M6 case.

Although the RAE 2822 case was run successfully on all 352 compute nodes of the Paragon used in this study, the M6 case could not be run on more than 200 nodes. The authors suspect a problem with orphan cells, but are hindered in their investigation because 128 nodes is the maximum that may be run on this Paragon without special permission.

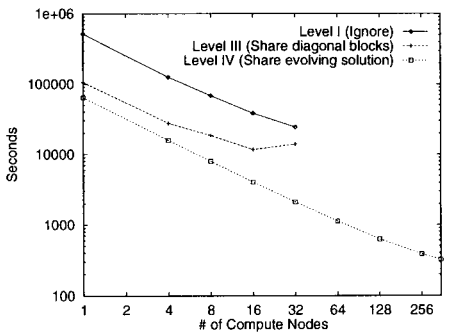


Figure 6: Time-to-converge for the RAE 2822 case.

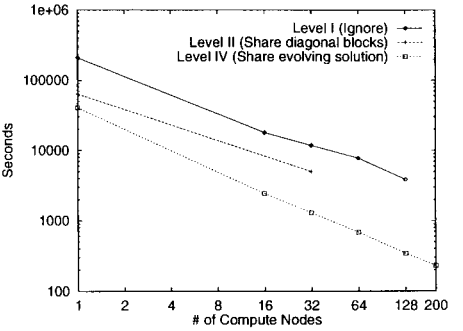


Figure 7: Time-to-converge for the ONERA M6 case.

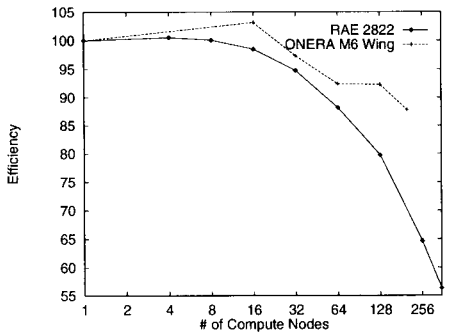


Figure 8: Efficiency for both cases using Level IV approximation (communication of evolving inner problem solution).

Also shown are figures displaying the clock time required to converge to a solution as well as the parallel efficiency of each of the schemes as a function of the number of compute nodes dedicated to the problem.

Finally, some problem-size dependency is illustrated in Figure 8, which shows efficiency results for both cases using Level IV approximation.

5. CONCLUSIONS

In spite of the large communications overhead associated with Level IV approximation, the time-to-converge is superior to all other schemes on the Paragon. This is because the convergence behavior on any number of compute nodes is almost identical to the behavior on one compute node, permitting utilization of very large CFL numbers.

Also, due to the Paragon's high-speed backplane, the communications costs were very low, contributing to the high efficiency of this scheme.

6. ACKNOWLEDGMENTS

This work was supported in part by a grant of High-Performance Computer (HPC) time from the DoD HPC Major Shared Resource Center Intel Paragon at Wright-Patterson Air Force Base, Dayton, Ohio. This work was also supported by the Office of Naval Research through the In-house Laboratory Independent Research (ILIR) program.

REFERENCES

[1] van Leer, B., "Towards the Ultimate Conservative Difference Scheme. V. A Second Order Sequel to Godunov's Method", *J. Comp. Phys.*, Vol. 32, 1979, pp.101-136.

[2]Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes", *J. Comp. Phys.*, Vol. 43, pp. 357-372.

[3]Karamcheti, K., *Principles of Ideal-Fluid Aerodynamics*, Krieger, Malabar, FL, 1980, p. 131.

[4]van Leer, B., "Towards the Ultimate Conservative Difference Scheme. IV. A New Approach to Numerical Convection", *J. Comp. Phys.*, Vol. 23, 1977, pp.276-299.

[5]Stoer, J. and R. Bulirsch, *Introduction to Numerical Analysis*, Springer-Verlag, New York, 1980, pp. 536-562.

[6]Sweby, P. K., "High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws", *SIAM J. Numer. Anal.*, Vol. 21, 1984, pp. 995-1011.

[7]Barth, T. J., and P. O. Frederickson, "Higher Order Solution of the Euler Equations on Unstructured Grids Using Quadratic Reconstruction", *28th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, 1990. AIAA 90-0013.

[8]Aftosmis, M., D. Gaitonde, and T. S. Tavares, "On the Accuracy, Stability, and Monotonicity of Various Reconstruction Algorithms for Unstructured Meshes", *32nd AIAA Aerospace Sciences Meeting and Exhibit*, Reno, 1994. AIAA 94-0415.

[9]van Albada, G. D., B. van Leer, and W. W. Roberts, "A Comparative Study of Computational Methods in Cosmic Gas Dynamics", *Astron. and Astrophys.*, Vol. 108, pp. 76-84.

[10]Venkatakrishnan, V., "On the Accuracy of Limiters and Convergence to Steady State Solutions", *31st AIAA Aerospace Sciences Meeting and Exhibit*, Reno, 1993. AIAA 93-0880.

[11]Duff, I. S., A. M. Erisman, and J. K. Reid, "Ordering Sparse Matrices to Special Forms", *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, 1986, pp. 153-157.

[12]Cook, P. H., M. A. McDonald, and M. C. P. Firmin, "Aerofoil RAE 2822 — Pressure Distributions, and Boundary Layer and Wake Measurements", *Experimental Data Base for Computer Program Assessment*, AGARD AR-138, 1979.

[13]Schmitt, V., and F. Charpin, "Pressure Distributions on the ONERA M6-Wing at Transonic Mach Numbers", *Experimental Data Base for Computer Program Assessment*, AGARD AR-138, 1979.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Analysis of Efficiency of Implicit CFD Methods on MIMD Computers

M. Perić and E. Schreck [a]

[a] Institut für Schiffbau, Universität Hamburg,
Lämmersieth 90, D-22305 Hamburg, Germany

The simulation of complex fluid flow problems requires huge amount of memory and long computing times. MIMD multiprocessors promise a scaleable arithmetic performance which can help to overcome the limitations in computing power and CPU-Memory bandwidth.

We consider the parallelization of an implicit numerical method for the prediction of flows in complex geometries with respect to MIMD computer architectures. During the last 20 years very efficient numerical methods have been developped for solving such problems on single processor machines. The major objective of our parallelization strategy is to preserve the high numerical efficiency of these sequential methods in the parallel implementation (i.e. to avoid performance degradation through domain decomposition) and to reduce the communication overhead (i.e. to achieve high parallel efficiency).

One possibilty to increase the parallel efficiency of implicit methods is to perform the communication in parallel with the calculation. Because of the hardware features of some new parallel computers (e.g. Intel PARAGON or Parsytec GCPP), which have special communication processors to allow overlapping of communication and computation on one node, this is a very promising method to hide or to minimize communication time.

1. Description of Solution Method

The solution method used in this study is described in detail by Demirdžić *et al.* [2], so only a summary of main features will be given here. The method is of finite volume type and uses non-orthogonal boundary-fitted grids with a colocated arrangement of variables. The working variables are the cartesian velocity components, pressure and temperature. The continuity equation is used to obtain a pressure-correction equation according to the SIMPLE algorithm (Patankar and Spalding,1972). Second order discretization is used for all terms. The part of diffusion fluxes which arises from grid non-orthogonality is treated explicitly. The convective fluxes are treated using the so called "deferred correction" approach: only the part which corresponds to the first order upwind discretization is treated implicitly, while the difference between the central differencing and upwind fluxes is treated explicitly.

Equations for the cartesian velocity components U and V, pressure correction P' and temperature T are discretized and solved one after another. Linear algebraic equation systems are solved iteratively using the ILU-decomposition (SIP) after Stone [5]. Inner iterations are stopped either after reducing the absolute sum of residuals over all CVs by

a specified factor, or after a prescribed number of iterations has been performed. Outer iterations are performed to take into account the non-linearity, coupling of variables and effects of grid non-orthogonality; this is why the linear equations need not be solved accurately.

The number of outer iterations increases linearly with the number of CVs, leading to a quadratic increase in computing time. For this reason a multigrid method is implemented, which keeps the number of outer iterations approximately independent of the number of CVs. The method is based on the so called "full approximation scheme" (FAS). It is implemented in the so called "full multigrid" (FMG) fashion. The solution is first obtained on the coarsest grid using the method described above. This solution provides initial fields for the calculation on the next finer grid, where the multigrid method using V-cycles is activated. The procedure is continued until the finest grid is reached. The coarse grids are subsets of the finest grid; each coarse grid CV is made of four CVs of the next finer grid. The equations solved on the coarse grids within a multigrid cycle contain an additional source term which describes the current solution and the residuals of the next finer grid. The multigrid method used in this study is described in detail in Hortmann *et al.* [3]. It should be noted that the multigrid method is applied only to the outer iteration loop; inner iterations are performed using the above mentioned solver irrespective of the grid fineness. This is due to the fact that the linear equations need not be solved accurately, so only a few inner iterations are necessary.

2. Parallelization Strategy and Efficiency

2.1. Grid Partitioning Technique

Domain decomposition technique is the basis of the parallelization strategy used in the present study. It is analogous to the block-structuring of grid in complex geometries. The aim is to have as many subdomains of approximately the same size as there are processors, and assigning each subdomain to one processor. More than one block may form a subdomain assigned to one processor. The subdomains do not overlap, i.e. each processor calculates only variable values for CVs within its subdomain. However, each processor uses some variable values from neighbour subdomains which are calculated by other processors. This requires, in case of MIMD computers with distributed memory, an overlap of storage: each processor stores data from one or more layers of CVs along its boundary belonging to neighbour subdomains. These data must be exchanged between processors.

In the SIMPLE algorithm, the variable values needed to calculate the coefficients and source terms are taken from the previous iteration. Therefore, they can be calculated in parallel. The equation system is split into subsystems, one for each subdomain, and these smaller systems are relaxed separately. Of course, this decreases the convergence rate, but it offers more flexibility and in most cases yields a shorter computing time than global parallelization of the single domain solver. The boundary data is exchanged after each inner iteration. This communication is *local* and can be performed in parallel. Some *global* communication is also needed, e.g. to sum the residuals for convergence check. The residual sums of all subdomains have to be collected, and the decision whether to stop or go has to be broadcast to all processors. This is normally done after each outer iteration,

and – unless a fixed number of inner iterations is prescribed – after each inner iteration. In the present algorithm, pressure is kept fixed at one node, but pressure correction is allowed to float; therefore, the pressure correction value at the reference node has to be subtracted from values at all other nodes. This requires broadcasting of the reference pressure correction value, which is done once per outer iteration. Flow diagramms of the inner and outer iterations are shown in Fig. 1, where the local (LC) and global (GC) communications are also indicated.

2.2. Efficiency of Parallel Implementation

The effectiveness of parallel computing can be characterized by the total efficiency, defined as the ratio of computing time on one processor using the most efficient serial algorithm, T_s, and the n-fold computing time using the parallelized algorithm and n processors, T_n:

$$E_n^{tot} = \frac{T_s}{nT_n} = E_n^{par} E_n^{num} E_n^{lb} . \tag{1}$$

Schreck and Perić [4] have shown that the total efficiency can be expressed as a product of three factors termed *parallel* (E_n^{par}) , *numerical* (E_n^{num}) and *load balancing* (E_n^{num}) efficiency. These factors describe: *(i)* the increase of elapsed time for a parallel computation due to communication between processors during which computation can not take place, *(ii)* the increase in the number of floating point operations per grid node required to reach the solution of the same accuracy when the number of subdomains is increased, and *(iii)* idle time of some processors due to uneven load.

The analysis by Schreck and Perić [4] shows that, especially for large processor sets and multigrid methods, the global communication has a strong effect on parallel efficiency. This is due to the fact that the time needed for one global communication is independent of the grid size.

One possibility for optimizing the efficiency is the hiding of communication time by using non-blocking communication. This is a hardware feature of some newer parallel computers and therefore a promising way.

The non-blocking local communication can be used in a quite straightforward way. In nearly all available message-passing libraries such communication modi are available. The effect is appreciable if there are sufficient arithmetic operations between sending and receiving of the boundary data. If grid partitions are sufficiently large, the communication time can be hidden completely. In most cases the latency time of the non-blocking communication is smaller than that of the blocking communication (on the computer used in this study , the ratio is 1 : 2). The overlap of local communication and computation is achieved by executing operations which require boundary data in seperate loops: between sending and receiving the boundary data, operation in the inner region are performed.

The restructuring of the loops should be done carefully because this can influence the caching behaviour of the data and therefore affect the processor performance.

The global communication offers an even higher saving potential because there are several nearest neighbour communications involved in every global operation. On the other hand it is not as easy as for the local communication to execute the global operations concurrently with the calculation. In this study the following strategy was adopted. Each processor spawn a communication thread which is connected to the computing thread

SIMPLE:

```
repeat
  assemble coefficient matrices for velocities
  solve linear equations
    LC: exchange A_P
  assemble coefficient matrices for pressure correction
  solve linear equations
    GC: broadcast reference pressure
  correct velocities and pressure
    LC: exchange corrected velocities
until converged
```

SIP:

```
assemble L and U matrices
repeat
  calculate residuals in inner domain
    LC: receive boundary data
  calculate boundary contribution to residuals
    GC: Send local part of residual for global sum
  calculate corrections
  correct variables on boundary
    LC: send boundary data
  Correct variable in inner domain
    GC: receive global residual sum
until converged
```

Figure 1. Pseudo Code for outer iterations (above) and for the SIP-solver (below) with non-blocking communication

with a common memory region. These threads on different processors are connected in an optimal manner for global operations (i.e. binary tree) and perform their communications synchronously. If there is need to handle several global operations simultaneously in the background one can replicate this strategy and spawn several communicating threads on each processor.

An important task remains in seperating the receiving of the result of the global operation as much as possible from the sending because this is the time the algorithm is accelerated. Only for a complete overlapping of global communication and calculation will the algorithm be scalable.

The time T_{glob} needed for a global operation can be estimated in the following way. In most cases only a small amount of data has to be transferred, therefore just the setup-time T_{setup} has to be considered:

$$T_{glob} = 2 \cdot 2 \cdot T_{setup} \cdot (ln(n) - 1), \tag{2}$$

where n is the number of processors. One factor of two is due to the fact that there is a reporting to the root of the binary tree and a broadcasting down the tree which require the same number of communications. In each stage there are two communications. This formula is strictly valid only in cases where a direct mapping of the tree on the processor network is possible. However, for most modern parallel computers there is only a slight increase in the setup time if there are additional hops needed in the communication. In this case Eq. (2) is still a good estimate. Typically the possible gain will be proportional to the number of arithmetic operations per CV, the number of CVs and the processor speed. Therefore, a certain minimum size of the subdomains is necessary for efficiently increase by using of non-blocking global operations.

The most critical part of the algorithm with respect to parallel efficiency is the solver of the linear systems. This is because of the fine granularity of the parallel linear solvers. Therefore we first analyze the possible gain from non-blocking communication in two typical linear solvers. For this purpose a matrix equation, originating from a discretized laplace-equation (which resembles the matrices in the CFD application), was solved . The stated computing times and efficiencies are related to one solver iteration thus reflecting the parallel efficiency. All test calculations presented in the following sections were performed on a Parsytec GC/PowerPlus consisting of PowerPC 601 microprocessors, running at 80MHz. The first solver used is a modified ILU decomposition after Stone [5]. Global communication is necessary to calculate the residual sum for steering the iteration process. It is possible to lag the checking of convergence one or more iterations, in which case a typical convergence rate has to be taken into account. The pseudo code for the ILU-solver with non-blocking communication is presented in fig. 1.

In Fig. 2 the measured computing times per solver iteration for three different grids are shown as a function of the processor number on an log-log plot . The ideal case with linear speedup should be a straight line in this plot. Especially the coarser grids show a clear deviation from this ideal case. The main reason for this behaviour is the high setup-time compared to the arithmetic performance ($T_{setup} = 1.5 \cdot 10^3 T_{flop}$). The lower setup-time in the non-blocking case is the main reason for the lower computing time on the coarser grids and fewer processors. The global communication exhibits a major effect on computing times for a higher number of processors. For too coarse grids it is not possible to hide the time for global communication completely, so the highest saving is obtained on the finest grid. In this case the run time with non-blocking communication shows nearly ideal behaviour.

In Fig. 2 the parallel efficiencies are shown for two different grids. For the finer grid and non-blocking (async.) communication the efficiency shows for non-blocking case only a weak decrease. Therefore, we could expect a good scaling behaviour with this approach up to a high number of processors. To verify this the load (i.e. the number of CVs) per processor was kept constant, so the size of the complete grid increased with the number of processors.

In Tab. 1 the measured computing times for two different loads and solvers are presented. The fact that on one processor higher computing times result for the non-blocking version is due to the loop splitting, which affects the cache-efficiency and hence the processor performance. However, the deterioration lais in the range of a few percent and will be compensated through gain from the non-blocking communication. In all cases an

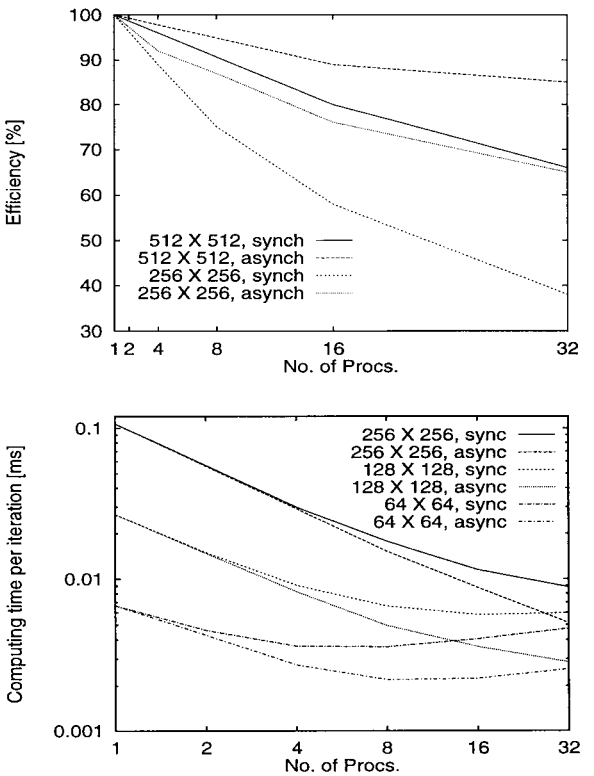


Figure 2. Efficiency (above) and computing times (below) for blocking and non-blocking version of the ILU-solver for different grid sizes

increase of the computing time can be observed up a configuration of 4×4 processors. This is due to the fact that the number of neighbours for the local communication reaches its maximum (4 in the 2D case) for the 4×4 configuration and remains constant from that point. A further doubling of the grid size and number of processors results in constant computing time for the non-blocking algorithm. This is opposed to the blocking algorithm, for which an increase in computing time is measured as a consequence of the blocking global communication.

As a second typical linear solver, a preconditioned conjugate gradient method was examined. In this case the global operations are necessary for computimg global scalar products. According to a proposal of Demmel et al. [1], it is possible to write the preconditioning Matrix M as $M = LL^T$. The scalar product of the residual r_k and the preconditioned residual $z_k = M^{-1}r_k$ can then be written as

$$(r_k, z_k) = (r_k, L^{-T}L^{-1}r_k) = (L^{-1}r_k, L^{-1}r_k) \ .$$

Therefore, the partial sums of (r_k, z_k) can already be calculated after the forward substitution so that the global sum can be overlapped with the backward substitution. In

addtion to that the update of the solution vector x_k is delayed in order to overlap it with the global reduction of the second scalar product (Ap_k, p_k), where p_k stands for the search vector.

For this solver there are just a few floating point operations per grid node which can be done during the overlap with the global communication (5 resp. 2 arithmetic operation per CV). As a consequence the measured computing times (Tab. 1) do not show the ideal behaviour as for the ILU-solver. Still there is a substantial gain for the non-blocking algorithm, especially for a higher number of processors.

	SIP				ICCG	
	64^2 per proc.		128^2 per proc.		128^2 per proc.	
# Proc.	block.	non-block.	block.	non-block.	block.	non-block.
1	6.8	7.4	27.2	28.4	60.6	60.5
2×2	9.1	8.2	30.0	29.3	63.9	61.8
4×4	11.4	8.8	33.3	30.3	68.5	62.6
4×8	12.8	8.9	35.4	30.5	72.6	63.8

Table 1
Computing times per iteration for the blocking and non-blocking version of the ILU and ICCG solver for different number of processors and different load per processor

3. Test Calculations and Analysis of Performance

The lessons learnt from the solver tests presented above are used to optimize the CFD-code. Buoyancy driven flow in an inclined cavity is used as a test case. This is one of the benchmark test cases presented by Demirdžić *et al.* [2]. All cavity walls are of the same area: the horizontal walls are adiabatic and the inclined walls (angle 45°) are isothermal. The temperature difference, cavity size and fluid properties are chosen such that for Prandtl number $Pr = 0.1$ the Rayleigh number $Ra = 10^6$ is obtained. Calculations were performed on 6 grid levels starting with 10×10 CV and up to 320×320 CV. In the

lag / (iter.)	time/(s)
orig	57.0
0	46.0
1	39.8
2	38.1
3	37.6

Algorithm	time/(s)	Total Eff. (%)
orig.	44.8	22
Mod. 1	30.5	33
Mod. 2	27.2	37

Table 2
Computing times for the described example on 5×5 processors using different number of iterations delay for the global communication in the solver (left) and different modifications to the algorithm (right)

original algorithm, see Fig. 1, all communication takes place in a blocking mode. In the modified algorithm, non-blocking communication is used. In addition to overlapping local communication with computing in the inner region, several variants of the global communication overlapping with computation were tested, since global communication is the limiting factor for massive parallelization. In particular, global communications for convergence check are overlapped with iterations. In order to evaluate the effect of non-blocking communication in the solver, the number of iterations was first fixed, so that global communication only affect the communication time and parallel efficiency. The effect of various delays between begin and end of global communication within inner iterations on overall computing time is shown in Table 2. A delay of 0 iterations means that only backward substitution and solution update are calculated during the global communication. Table 2 shows that the major gain is due to introducing non-blocking communication and using up to one iteration delay. With respect to computing time, there is only a small advantage from using more than 2 iterations delay. The reason is that on finer grids, the computing time outweighs the communication time with only very few iterations delay.

Further tests were performed by switching the convergence check for inner iterations on. A delay of two iterations was used and the convergence criterion for the inner iterations was adapted according to the convergence rate to respect the fact that the residual values two iterations old are used for checking. This is called modification 1 in Tab. 2. In the next step (modification 2), the non-blocking mode was used for all global communications. The gain for the last optimization is less than that for the first modification. As mentioned earlier, this is due to the fact that the frequency of communications in the linear solver part is much higher than for assembling of the matrices. The saving in computing time is substantial. Total efficiencies are rather low on computer used, due to the effect of communications on coarse grids. When the first two grids are left out, the efficiency of modification 2 increases from 37% to 54% with 5×5 processors (70% with 4×4 processors). On transputer systems, which have a favourable ratio of communication to computing time, blocking communication leads to efficiencies of over 90% (Schreck and Perić [4])

REFERENCES

1. Demmel, Heath and Van Der Vorst, Parallel numerical linear algebra, *Acta Numerica*, 1993, 111-197
2. I. Demirdžić, Ž. Lilek and M. Perić: "Fluid flow and heat transfer test problems for non-orthogonal grids: benchmark solutions", *Int. J. Num. Methods in Fluids*, **15**, 329-354 (1992).
3. M. Hortmann, M. Perić and G. Scheurer: "Finite volume multigrid prediction of laminar natural convection: bench-mark solutions", *Int. J. Numer. Methods Fluids*, **11**, 189-207 (1990).
4. E. Schreck and M. Perić: "Computation of fluid flow with a parallel multi-grid solver", *Int. J. Num. Methods in Fluids*, 16, 303 - 327 (1993).
5. H.L. Stone: "Iterative solution of implicit approximations of multi-dimensional partional differential equations", *SIAM J. Numer. Anal.*, **5**, 530-558 (1968).

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel Solutions of Three-Dimensional Compressible Flows Using a Mixed Finite Element/Finite Volume Method on Unstructured Grids

S. Lanteri[a] and M. Loriot[b]

[a]INRIA, 2004 Route des Lucioles, B.P. 93, 06902 Sophia-Antipolis Cedex, France

[b]Simulog, 1 Rue James Joule, 78286 Guyancourt, France

We are concerned here with the MIMD parallelisation of an existing industrial code, N3S-MUSCL (a three-dimensional compressible Navier-Stokes solver, see Chargy[1]). Part of the work described in this paper is supported by the CEC through consortium HPCN3S of the Europort-1 project.

1. INTRODUCTION

The porting methodology of a given serial algorithm to a parallel machine generally depends on the characteristics of the selected target parallel architecture, but also on some maintenance requirements. As an example, in some of our previous works (see Fezoui *et al.*[2], Farhat *et al.*[3]), our main goal was to achieve a maximum level of performance on a particular machine (the Tmc CM-2/200), thus accepting to introduce significant modifications to the original serial algorithm. This approach may be of interest for "simple" demonstrations codes which generally deal with the solution of one or two-dimensional problems. However, for *industrial* codes, such as the multi-purpose CFD code described here, a strong requirement is to minimize the changes to the original serial algorithm in order to be able to maintain and upgrade the resulting parallel version of the code.

The main characteristic of N3S-MUSCL is that it is based on finite volume schemes using finite element type grids (tetrahedra), which results in complex data structures. The conservative form of the Navier-Stokes equations is discretised using a mixed finite element/finite volume method on fully unstructured meshes. The parallelisation strategy adopted in this study combines mesh partitioning techniques and a message-passing programming model. The mesh partitioning algorithms and the generation of the corresponding communication data-structures are gathered in a preprocessor in order to introduce a minimum change in the original serial code. Two partitioning approaches have been implemented and compared here. The first one makes use of overlapping mesh partitions; it contributes to minimize the programming effort on the original serial algorithm but is characterized by redundant arithmetic operations. This work is undertaken in the HPCN3S project. The second strategy, developped at INRIA Sophia-Antipolis in a research programme, uses non-overlapping mesh partitions and demands additional programming effort.

2. THE FEM/FVM NAVIER-STOKES SOLVER

Here, we briefly overview the spatial and temporal discretisation methods that are detailed in Chargy[1] for the numerical solution of the full three-dimensional Navier-Stokes equations.

2.1. Governing equations

Let $\Omega \subset I\!R^3$ be the flow domain of interest and Γ be its boundary. The conservative law form of the equations describing three-dimensional Navier-Stokes flows is given by:

$$\frac{\partial W}{\partial t} + \vec{\nabla}.\vec{\mathcal{F}}(W) = \frac{1}{Re}\vec{\nabla}.\vec{\mathcal{R}}(W) \tag{1}$$

where $W = (\rho, \rho\vec{U}, E)^T$; $\vec{\mathcal{F}}(W)$ is the vector of convective fluxes while $\vec{\mathcal{R}}(W)$ is the vector of diffusive ones. In the above expressions, ρ is the density, $\vec{U} = (u, v, w)^T$ is the velocity vector, E is the total energy per unit of volume. Finally, $Re = \rho_0 U_0 L_0/\mu_0$ where ρ_0, U_0, L_0 and μ_0 denote the characteristic density, velocity, length, and diffusivity is the Reynolds number.

2.2. Spatial approximation method

The flow domain Ω is assumed to be a polyhedral bounded region of $I\!R^3$. Let $\mathcal{T}_h$ be a standard tetrahedrisation of Ω, and h the maximal length of the edges of $\mathcal{T}_h$. A vertex of a tetrahedron T is denoted by S_i, and the set of its neighbouring vertices by $K(i)$. At each vertex S_i, a control volume C_i is constructed as the union of the subtetrahedra resulting from the subdivision by means of the medians of each tetrahedron of $\mathcal{T}_h$ that is connected to S_i (see Fig. 1). The boundary of C_i is denoted by ∂C_i, and the unit vector of the outward normal to ∂C_i by $\vec{\nu}_i = (\nu_{ix}, \nu_{iy}, \nu_{iz})$. The union of all these control volumes constitutes a discretisation of domain Ω:

$$\Omega_h = \bigcup_{i=1}^{N_V} C_i \quad , \quad N_V : \text{number of vertices of } \mathcal{T}_h \tag{2}$$

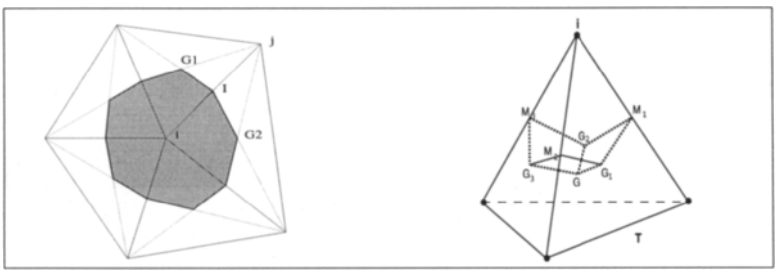


Figure 1. 2D control surface (left) and contribution to a 3D control volume (right)

The spatial discretisation method adopted here combines a finite volume upwind approximation method for the convective fluxes and a classical Galerkin finite element centered approximation for the diffusive fluxes. The numerical convective fluxes are computed using the approximate Riemann solver of Roe[5]. Second-order spatial accuracy is achieved by using an extension to unstructured meshes of the "Monotonic Upwind Scheme for Conservative Laws" (MUSCL) method introduced by van Leer [6]; The discrete equation obtained after integrating Eq. (1) over C_i is given by:

$$\begin{aligned} \iiint_{C_i} \frac{\partial W}{\partial t} d\vec{x} \; + \; \sum_{j \in K(i)} \int_{\partial C_{ij}} \vec{\mathcal{F}}(W).\vec{\nu}_i d\sigma \; + \int_{\partial C_i \cap \Gamma_w} \vec{\mathcal{F}}(W).\vec{n}_i d\sigma \\ + \int_{\partial C_i \cap \Gamma_\infty} \vec{\mathcal{F}}(W).\vec{n}_i d\sigma \quad = -\frac{1}{Re} \sum_{T, S_i \in T} \iiint_T \vec{\mathcal{R}}(W).\vec{\nabla} N_i^T d\vec{x} \end{aligned} \tag{3}$$

where $\partial C_{ij} = \partial C_i \cap \partial C_j$, and $N_i^T = N_i^T(x, y, z)$ is the P1 shape function defined at the vertex S_i and associated with the tetrahedron T. We refer to [1] for a detailed description of the computation of each of the terms of Eq. (3).

2.3. Time Integration

Assuming that $W(\vec{x}, t)$ is constant over the control volume C_i (in other words, a mass lumping technique is applied to the temporal term of Eq. (3)), we obtain the following semi-discrete fluid flow equations:

$$\mathrm{vol}(C_i) \frac{dW_i^n}{dt} + \Psi(W_i^n) = 0 \quad , \quad i = 1, \cdots, N_V \tag{4}$$

where $W_i^n = W(\vec{x}_i, t^n)$, $t^n = n\Delta t^n$ and:

$$\Psi(W_i^n) = \sum_{j \in K(i)} \Phi_{\mathcal{F}}(W_{ij}, W_{ji}, \vec{\nu}_{ij}) + \int_{\partial C_i \cap \Gamma_\infty} \vec{\mathcal{F}}(W).\vec{n}_i d\sigma + \frac{1}{Re} \sum_{T, S_i \in T} \vec{\mathcal{R}}_i(T) \tag{5}$$

The time advancing procedure considered in this study relies on an implicit linearised formulation following the work of Fezoui and Stoufflet[7]. The resulting scheme is in fact a modified Newton method involving an approximate Jacobian :

$$P(W^n) \delta W^{n+1} = \left(\frac{I}{\Delta t^n} + J(W^n) \right) \delta W^{n+1} = \delta \hat{W}^n \quad , \quad \delta W^{n+1} = W^{n+1} - W^n \tag{6}$$

where $J(W^n)$ denotes the approximate Jacobian matrix and $\delta \hat{W}^n$ is the explicit part of the linearisation of $\Psi(W^{n+1})$. The matrix $P(W^n)$ is sparse and has the suitable properties (diagonaly dominant in the scalar case) allowing the use of a relaxation procedure (Jacobi or Gauss-Seidel) in order to solve the linear system of Eq. (6). Moreover, an efficient way to get second order accurate steady solutions while keeping the interesting properties of the first order upwind matrix is to use a second order elementary convective flux in the right-hand side of Eq. (6).

3. PARALLEL IMPLEMENTATION ISSUES

The parallelisation strategy adopted in this study has been already successfully applied in the two-dimensional case (see Fezoui *et al.*[2], Farhat *et al.*[4]); preliminary results for three-dimensional applications are presented in Degrez *et al.*[8]. It combines mesh partitioning techniques and a message-passing programming model. The underlying mesh is assumed to be partitioned into several submeshes, each defining a subdomain. Basically the same "old" serial code is going to be executed within every subdomain. Modifications occured in the main time-stepping loop in order to take into account one or several assembly phases of the subdomain results, depending on the order of the spatial approximation and on the nature of the time advancing procedure (explicit/implicit). The assembly of the subdomain results can be implemented in one or several separated modules and optimized for a given machine. This approach maximises the portability of the resulting code.

The reader can verify that for the mixed finite volume/finite element formulation considered herein, mesh partitions with overlapping simplify the programming of the subdomain interfacing module. However, mesh partitions with overlapping also have a drawback: they incur redundant floating-point operations. On the other hand, non-overlapping mesh partitions incur little redundant floating-point operations but induce additional communication steps. While physical state variables are exchanged between the subdomains in overlapping mesh partitions, partially gathered nodal gradients and partially gathered fluxes are exchanged between subdomains in non-overlapping ones. In addition, special care must be taken in the treatment of the convective fluxes in the case of non-overlapping mesh partitions (because of the possible differences in the orientation of the interface edges which are not part of the original mesh but are instead constructed during a preprocessing phase of the parallel algorithm). In other words, the programming effort is maximized when considering non-overlapping mesh partitions. In the present study we will consider both one tetrahedron wide overlap and non-overlapping mesh partitions for second order accurate implicit computations.

For the time integration procedures considered in this study, an automatic mesh partitioner should focus primarily on creating load balanced submeshes which induce a minimum amount of interprocessor communications. This can be achieved by using a two-step procedure. First, a fast and cheap partitioning scheme is used to derive an initial candidate; then, an optimisation process is performed in order to realise the stated goals. While the former step consists in a global operation (the overall mesh is concerned by this step), the latter mainly concentrate on those mesh components that are neighbours of the artificial submesh interfaces (local operation). Optimisation techniques that are used in this context include (among others) simulated annealing and the Kernighan-Lin algorithm. Here, the computational mesh is partitioned in a preprocessing step. We have used two special purpose packages that implement several mesh partitioning algorithms : `MS3D` (a Mesh Splitter for 3D applications, see Loriot[9]) for the construction of one tetrahedra wide overlapping mesh partitions using a recursive inertia bisection algorithm, and `TOP/DOMDEC` (a software tool for mesh partitioning and parallel processing of CSM and CFD computations [10]) to generate non-overlapping partitions using a Greedy algorithm.

4. PERFORMANCE RESULTS

The test case we consider is the one of the Euler flow around an **ONERA M6** wing. The angle of attack is set to 3.06° and the free stream Mach number to 0.84; we present results of simulations performed with a mesh containing 115,351 vertices and 643,392 tetrahedra. Fig. 2 visualises the steady Mach lines and clearly shows the λ-shock pattern on the wing surface.

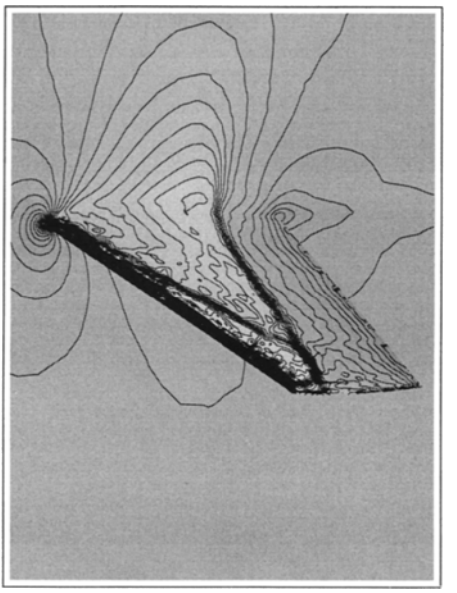


Figure 2. Steady Mach lines on an **ONERA M6** wing : $N_V = 115,351$ - $N_T = 643,392$

Timing measures concern the main parallel loop. Unless stated otherwise, the reported CPU times always refer to the maximum of the individual processor measures. In the following tables N_p is the number of involved processors (submeshes) while "Loc Comm" and "Glb Comm" respectively denote the local (send/receive at artificial submesh boundaries) and global communication times. In the case of local communication operations, the corresponding measures include the time spent in packing and unpacking message buffers. The parallel speed-up has been approximated as:

$$S(N_p) = \frac{T_1^s}{T_{N_p}^p} = \frac{N_p \times T_{comp}^p}{T_{comp}^p + T_{comm}^p} = N_p \times \left(\frac{1}{1 + \frac{T_{comm}^p}{T_{comp}^p}}\right) \tag{7}$$

where T_{comm}^p and T_{comp}^p respectively denote the computation time and the communication time of the parallel application.

We first report results obtained with a simplified version of the code N3S-MUSCL which is able to deal with overlapping as well as non-overlapping mesh partitions. The implicit time advancing procedure is used with 36 Jacobi relaxations for the approximate solution of the linear system resulting from (6). The pseudo-time step is computed according to the law CFL=4 $\times$ *it* where *it* denotes the current non-linear iteration. The steady state solution (initial normalized residual divided by 10^6) has been obtained after 196 non-linear iterations. Tab. 1 compare the total CPU times for parallel solution algorithms based on overlapping ("Size = 1") and non-overlapping ("Size = 0") mesh partitions. All performance results reported below are for 64-bit arithmetic.

Table 1
Implicit Euler computations on the ONERA M6 : $N_V = 115,351$ - $N_T = 643,392$
Computations on the Intel Paragon : NX communication library

N_p	Size	CPU	Mflop/s	Loc Comm		Glb Comm		$S(N_p)$
				Min	Max	Min	Max	
64	1	4348.0 s	223	45.5 s	98.0 s	8.0 s	9.5 s	62
	0	2819.0 s	344	57.0 s	126.0 s	8.0 s	10.0 s	61
128	1	2824.0 s	345	25.0 s	66.0 s	9.0 s	11.5 s	124
	0	1572.0 s	619	36.0 s	103.0 s	9.5 s	12.0 s	118

Computations on the Cray T3D : PVM communication library

N_p	Size	CPU	Mflop/s	Loc Comm		Glb Comm		$S(N_p)$
				Min	Max	Min	Max	
64	1	1464.0 s	664	30.0 s	85.5 s	14.5 s	17.5 s	60
	0	1138.0 s	855	38.0 s	153.0 s	14.5 s	16.0 s	54

Computations on the Ibm SP-2 (wide nodes) : MPL communication library

N_p	Size	CPU	Mflop/s	Loc Comm		Glb Comm		$S(N_p)$
				Min	Max	Min	Max	
32	1	726.0 s	1340	29.0 s	43.0 s	4.0 s	5.0 s	30
	0	570.0 s	1705	33.0 s	50.0 s	3.5 s	4.5 s	29
64	1	444.0 s	2190	19.0 s	35.0 s	5.5 s	6.0 s	58
	0	316.0 s	3077	24.0 s	51.0 s	4.5 s	5.5 s	55

We remark that the estimated speed-up figures are always better when using overlapping mesh partitions. This behaviour is simply explained by the fact that between computations with overlapping and non-overlapping mesh partitions the pure computational times decrease while the communication times increase in such a way that the ratio between the two figures is not favorable to an improved speed-up. Indeed, the pure computational times for overlapping mesh partitions include redundant floating-point operations which means that larger global problems are actually solved in this case. This suggests that techniques for overlapping communication steps and purely computational ones should be investigated in order to improve the performances of the parallel algorithm based on non-overlapping mesh partitions. For the case $N_p = 64$ the total communication times for non-overlapping partitions ("Size = 0") are equal to 130.0 s on the Intel Paragon, 169.0 s on the Cray T3D and 56.5 s on the Ibm SP-2. These figures respectively represent 5%,

15% and 18% of the corresponding total CPU times; this explains the higher speed-up figures obtained on the Intel Paragon.

We now present results obtained with the *industrial* version of the N3S-MUSCL code which is currently ported on various parallel platforms in the context of the HPCN3S project. This time, we consider the multi-species (two chemical species) Euler flow around an ONERA M6 wing (the angle of attack is set to 3.06° and the free stream Mach number to 0.84). The underlying mesh contains 53,961 vertices and 287,962 tetrahedra. The calculations are made on an Ibm SP-2 (equipped with thin nodes) using overlapping mesh partitions (one tetrahedra wide). The implicit time advancing procedure is used with 30 Jacobi relaxations for the approximate solution of the linear system resulting from (6). The pseudo-time step is computed according to the law CFL=it where it denotes the current non-linear iteration. Tab. 2 below compares the CPU times per non-linear iteration; C_{comm} denotes the percentage of the CPU times spent in local communication operations. Here, computations have been performed using 32-bit arithmetic.

Table 2
Implicit Euler computations on the ONERA M6 : $N_V = 53,961$ - $N_T = 287,962$
Computations on the Ibm SP-2 (thin nodes) : PARMACS communication library

N_p	CPU/iter	C_{comm}	Mflop/s	$S(N_p)$
2	65.7 s	1.0 %	38	1.9
4	33.0 s	3.6 %	80	3.9
8	16.6 s	6.0 %	168	7.7
15	8.6 s	7.2 %	342	14.4

5. PERFORMANCE MODELLING

In this section, we describe a performance model for the *industrial* N3S-MUSCL solver. The model is implemented to be used inside the mesh partitioner MS3D, in order to provide the end-users with accurate prediction of the parallel code behaviour, for a given decomposition and a given architecture.

In the following we assume that one subdomain is assigned to each processor and that, in the parallel code, communication steps and purely computational ones do not overlapped. The partitioning of the mesh gives access to the following local (per subdomain) informations : for each message, the identifications of the sender and the receiver and the message length (number of interface vertices); the numbers of vertices and tetrahedra characterizing the local submesh. From the local mesh parameters of the ith subdomain and the numerical/physical parameters of the simulation (number of species, number of relaxations per iteration, type of solver, type of numerical flux, $\cdots$), we can derive a local computation time, T^i_{comp}. The communication time for sending a message is computed as $t_{comm} = t_{latc} + N * b$ where t_{latc} is the latency in seconds, N is the number of bytes to be sent and b is the bandwidth in Mbytes/s. These figures are those associated to the selected message passing library (PARMACS) instead of the ones of the targeted system, therefore including software overheads. By adding together all the various communication times for the ith subdomain, we get an overall local communication time, T^i_{comm}. The total

CPU time of one non-linear iteration for the ith subdomain is $T^i = T^i_{comm} + T^i_{comp}$. We can now compute the total CPU time and average computational time of one non-linear iteration on N_p processors from which we can deduce the average communication time T_{comm} (which actually include the idle time due to computational load imbalance) :

$$T = \max_{i=1,N_p} T^i \quad , \quad T_{comp} = \frac{1}{N_p} \sum_{i=1}^{N_p} T^i_{comp} \quad , \quad T_{comm} = T - T_{comp} \tag{8}$$

Several validation experiments have been realised using the described model. For instance, the previous `ONERA M6` wing test case (2,800 vertices and 13,576 tetrahedra) was run on 16 processors of a `Meiko CS-1` system. The data taken for this system (obtained courtesy of Pallas GmbH) were : $t_{latc} = 0.5$ ms, $b = 2$ Mbytes/s. The results have shown a difference of less that 3% (CPU/iter times were 7.42 s for the experiment and 7.6 s for the prediction). For larger problem sizes and larger sys tem configurations the obtained results all tend to indicate that the choice of the partitioning algorithm is not crucial, but the computational load balance is.

Acknowledgments : performance results obtained on large configurations of the `Ibm SP-2` and the `Intel Paragon` have been provided by Pr. C. Farhat (*Center for Aerospace Structures*, University of Colorado at Boulder). The first author also wishes to thank Mr. J.-M. Fieni for giving him access to the `Cray T3D` located at the CEA in Grenoble.

REFERENCES

1. D. Chargy, `N3S-MUSCL` : a 3D Compressible Navier-Stokes Solver, Edf/Simulog (1994).
2. L. Fezoui and S. Lanteri, Parallel Upwind FEM for Compressible Flows, Proceedings of the PCFD'91 Conference, K.G. Reinsch *et al.* Eds., pp. 149-163, (1992).
3. C. Farhat, L. Fezoui and S. Lanteri, Two-Dimensional Viscous Flow Computations on the CM-2: Unstructured Meshes, Upwind Schemes and Massively Parallel Computations, Comp. Meth. in Appl. Mech. and Eng., Vol. 102, pp. 61-88, (1993).
4. C. Farhat and S. Lanteri, Simulation of Compressible Viscous Flows on a Variety of MPPs: Computational Algorithms for Unstructured Dynamic Meshes and Performance Results, Comp. Meth. in Appl. Mech. and Eng., Vol. 119, pp. 35-60, (1994).
5. P.L. Roe, Approximate Riemann Solvers, Parameters Vectors and Difference Schemes, J. of Comp. Phys., Vol. 43, pp. 357-371, (1981).
6. B. Van Leer, Towards the Ultimate Conservative Difference Scheme V : a Second-Order Sequel to Godunov's Method, J. of Comp. Phys., Vol. 32, pp. 361-370, (1979).
7. L. Fezoui and B. Stoufflet, A Class of Implicit Upwind Schemes for Euler Simulations with Unstructured Meshes, J. of Comp. Phys., Vol. 84, pp. 174-206, (1989).
8. G. Degrez, L. Giraud, M. Loriot, A. Micelotta, B. Nitrosso and A. Stoessel, Parallel Industrial CFD calculations with N3S Proceedings of the HPCN'95 Conference, B. Hertzberger *et al.* Eds., Lect. Notes in Comp. Sc., Springer, Vol. 919, pp. 820-825, (1995).
9. M. Loriot, MS3D : Mesh Splitter for 3D Applications, User's Manual, Simulog, (1992).
10. C. Farhat, S. Lanteri and H. Simon, TOP/DOMDEC : a Software Tool for Mesh Partitioning and Parallel Processing and Applications to CSM and CFD Computations, Comput. Sys. in Engrg., (To Appear), (1995).

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Implementation of a Fractional Step Algorithm for the Solution of Euler Equations on Scalable Computers

A. G. Stamatis and K. D. Papailiou

Lab. of Termal Turbomachines,National Technical University of Athens ,P.O Box 64069, 157 10, Athens, Grece, Tel:(1)7722343,Fax:(1)7784582, email:stamatis@central.ntua.gr

AbstractThis paper deals with the parallel implementation of an Euler Explicit Fractional Step Solver on different scalable computing platforms, such as the Intel's PARAGON and Parsytec's GCel.

1. Introduction

Many efforts have been made during the last decade for using parallel computers to solve CFD problems in order to verify the effectiveness and suitability of such architectures, in particular when large scale problems have to be faced. Although a certain fraction of these efforts has been oriented to shared memory multiprocessors as well as to SIMD massively parallel architectures, it is fairly recognized that future CFD demands lead to the efficient exploitation of distributed memory scalable computers. Whether using finite differences, finite volumes or finite elements, efficient methods for solving the time dependent Euler equations typically involve two basic types of schemes: the explicit and the implicit.

Implicit methods present the advantage of allowing larger time steps compared to explicit ones, provided that an implicit treatment of the boundary conditions is incorporated, which in turn leads to a further complexity of the code structure. On uni-processor and vector architectures the implicit schemes have proven to be superior in many instances, but due to inherent global spatial data dependencies these methods are hard to parallelize. Sophisticated rearrangement of the computation steps and complex communication patterns are required in order to maintain efficiency of the implicit schemes on parallel system as well,[1].On the other hand, implementation of explicit algorithms on a distributed memory environment is anticipated (and has been proven) to be both easy and efficient because of inherently parallel nature of these algorithms.

This paper deals with the parallel implementation of an Euler Explicit Fractional Step Solver on different scalable computing platforms, such as the Intel's PARAGON and Parsytec's GCel.A certain advantage of using a fractional-step analysis, beyond its simplicity is the fact that greater time-steps are allowed, since the stability criterion is less strict compared to other explicit solvers.

2. Description of the Algorithm

Euler equations in two dimensions can be written in the following conservative

form

$$\frac{\partial \vec{Q}}{\partial t} + \frac{\partial \vec{F}}{\partial \xi} + \frac{\partial \vec{G}}{\partial \eta} = 0 \qquad (1)$$

where the unknown variable vector Q, and the flux vectors F,G one defined as usually and ξ,η are the computational domain coordinates.

The fractional-step concept implies that the finite-difference form of the discretized equation (1) split into a sequence of multiple single-directional operators,[2]. Each operator corresponds to a different physical component of the equation. The 1-D operators are denoted by L and are subscribed by either ξ or η, depending on whether sweeps are performed along the η or ξ=constant grid lines respectively.The time evolution of the unknown vector array Q is obtained by applying the sequence of operators, which may be cast in the following symbolic form

$$\vec{Q}^{n+2} = L_\xi L_\eta L_\eta L_\xi \vec{Q}^n \qquad (2)$$

A double and inverse sequence of the one-dimensional operators leads to a second order accuracy in time, while the calculated quantities have a physical meaning only at the expiration of a $2\Delta t$ time interval (i.e. from n to n+2 iteration level,[3]).The predictor-corrector MacCormack scheme is used to handle the hyperbolic operators, [4]. A common time-step is used for all L operators and this is the minimum of the time steps, which result from the stability analysis performed for each operator separately. The total time-increment per time-step is less strict compared to that dictated by any explicit two-dimensional stability criterion.

Extra dissipation terms are explicitly added to the solution vector Q at the end of a complete calculation period, corresponding to Q time interval of $2\Delta t$ as follows

$$\vec{Q}^{n+2} = \vec{Q}^{n+2} + \vec{D}_\xi + \vec{D}_\eta \qquad (3)$$

where $D_\xi+D_\eta$ represents a blend of second and fourth order derivatives of the solution vector following Jameson, [5], to prevent odd-even uncoupling and to accurately capture shock waves.

Quite a few test cases have been investigated using the above algorithm with appropriate boundary conditions.Details of the formulation and results can be found in [6-7].

3. Parallelization of the Algorithm

3.1 Methodology

Parallelization on different distributed memory computers has been introduced by a subdomain-partitioning strategy.The computational domain is decomposed into subdomains assigned to different processors. In order to update grid points that lie on the edge of its subdomain each processor needs values for the grid points which lie in an adjacent subdomain, and this is done by exchanging data. The decomposition of the domain can be 1-D (stripes) or 2-D (squares). Given that each decomposition has its own

advantages with respect to the processors utilized and the aspect ratio of computational domain, [8], we decided to provide the code with the capability of using either decomposition.
Thus, each processor must store the data for a number of rows depending of the stencil of grid points on each side of its region of interest.For the fractional step algorithm in each fractional step, data belonging to one row (or column) of a side of each subdomain has to be exchanged with the processor that is responsible for the next subdomain along the sweep direction. The only exception is during calculation of the artificial dissipation terms, where two rows (or columns) have to be exchanged.

The second aspect of the parallelization strategy is the mapping of subdomains to processors. The key feature of a suitable mapping is data locality (i.e. adjacent subdomains to be assigned to neighbouring processors in physical topology). Assuming that the underlying physical processor topology is a 2D mesh (Paragon XP/S, Parsytec GCel), we may have a direct one to one mapping. The exchange of the data between neighbouring processes is only local and can be performed in parallel on all processors. A different communication need is present when non-neihgbouring processors have to exchange data (wake region behind an airfoil). Communication between the processors is also necessary for monitoring of the residual values.

Portability remains at present time one major problem of parallel computing. Different MIMD architectures generally offer incompatible message passing and processor control possibilities. Though some "standards" (e.g. PVM,P4) begin to be implemented on virtually all major computers, here is generally a non negligible overhead when using these modules instead of the local native message passing libraries. The concept of code portability involves two requirements: a) the code should compile on different computers, b) if the code runs efficiently on one system should also run efficiently on other systems as well.

In order to achieve an easy and efficient portability, the flow solver is completely separated from the library of subroutines controlling the parallel work.Communication code consists of a main library developed in our Lab, and several communication primitive libraries.At present communication is based on an assumed 2-D processor topology. Mapping of the actual hardware topology to the logical 2-D processor topology is provided through a user defined subroutine (by default the underlying hardware topology is assumed to be 2-D mesh).With the aid of the above methodology a single source code can be executed either sequentially or in parallel. This greatly simplifies development and maintenance of the parallel code.

3.2 Parallel Performance Model

The issue of evaluating the performance of an application in a parallel environment has a very complicated nature and is characterized by a large number of often contradictory aspects. No matter how have been defined or interpretated, [9-11], various metrics of the parallel performance (speedup, efficiency, serial fraction, etc.), may give a practical view of the quality of the parallel implementation as well as of the usability of the parallel environment.

Should the target be the answer, which algorithm and what parallel computer are most appropriate for a given application, one has to consider a metric sensitive to the

characteristic parameters both of the algorithm and the parallel computer.We define such a metric named effective speed-up as follows

$$S_{ef} = \frac{T(plat_{ord}, alg_{ord}, P_{ord})}{T(plat, alg, P)} \tag{4}$$

where T is the execution time, plat identifies computer platform, alg the algorithm used and P the number of processors which are occupied by the application. The subscript ord refers to the ordinary implementation of the application. Clearly, when plat=$plat_{ord}$, alg=alg_{ord} and p_{ord}=1 then S_{ef} represents the traditional parallel speed-up. The use of the effective speed-up appears certain advantages. Using the same algorithm and the same number of processors (e.g. p=p_{ord}) one may have a comparison of the machines. Using a specified platform and the same number of processors we may compare different algorithms for the same application. But the major advantage of using the metric of effective speed-up is that the absolute gain obtained through an implementation different from the ordinary one ,can be evaluated.

In order to be able to investigate the expected parallel behaviour of an implementation in a target machine before the actual implementation, time complexities or parallel performance models are valuable tools. The parallel performance model used in the present approach is derived through the concept of Work Units [10]. We may consider that there are K discrete jobs of the algorithm during one iteration.Then assuming a linear message passing communication time, in the SPMD (single program, multiple data) paradigm adopted here, using p processors it can be derived the following approximation for the parallel time of the application

$$T(plat,alg,p) = n_{iter} \{K_f t_f + K_s t_s + K_t t_t\} \tag{5}$$

where

$$K_f = \sum_{i}^{k} \max_{j=1,p}\{f_{ij}\} \qquad K_s = \sum_{i}^{k} \max_{j=1,p}\{s_i\} \qquad K_t = \sum_{i}^{k} \max_{j=1,p}\{L_{ij}\} \tag{6}$$

and

t_f=t_f(plat) : Mean time to complete one floating point operation.
t_s=t_s(plat) : Setup communication time (Latency).
t_t=t_t(plat) : Time required to exchange 1 byte (is determined from processor to processor bandwidth).
f_{ij}=f_{ij}(alg,p) : Number of floating point operations of computational domain during Work Unit i.
s_i=s_i(alg) : Number of times data are exchanged during Work Unit i.
L_{ij}=L_{ij}(alg,p) : Message length exchanged by processor j during Work Unit i.
n_{iter} : Iterations required for convergence.

3.3 Algorithmic Alternatives

From the parallel performance model of the previous paragraph we may deduce the

impact of changing the algorithm in the expected speed-up. We will consider two approaches. The first one is to double calculations in order to reduce communication time. The expected speed-up for a given platform and number of processors is

$$S = \frac{n_{iter}^{1}\left\{K_f^1 t_f + K_s^1 t_s + K_t^1 t_t\right\}}{n_{iter}^{2}\left\{K_f^2 t_f + K_s^2 t_s + K_t^2 t_t\right\}} \tag{7}$$

Although the algorithm changes, the number of iterations required for convergence remains the same. Thus,given that for our implementation of the fractional step algorithm it holds that

$K_s^1 \approx K_s^2$, $(K_f^2-K_f^1)/(K_s^1-K_s^2) \approx (N/p)^{1/2}$

in order to ensure $S \geq 1$ the following condition must be fulfilled

$$\frac{t_t}{t_f} \geq \sqrt{(N/p)} \tag{8}$$

The conclusion drawn from the above selection is that the double calculation algorithm is efficient only for computers with very high communication time to calculation time ratio.

The second algorithmic alternative is to decrease communication time by exchanging information not every iteration but after a certain number of iterations. In that case $K_f^2=K_f^1=K_f$, $K_s^2=K_s^1/n$ and $K_t^2=K_t^2/n$, where n the number of iterations required to exchange messages. From (7) in order to obtain $S \geq 1$ we have the necessary condition

$$\frac{n_{iter}^{2}}{n_{iter}^{1}} \leq \frac{1+a}{1+a/n} \tag{9}$$

where

$$a = 1 + \frac{K_s t_s}{K_f t_f} + \frac{K_t t_t}{K_f t_f} \tag{10}$$

Clearly, there is no apriori answer if it is worth to apply the algorithm. Only after investigating the convergence rate in a particular machine for a given number of processors and for various n we will be able to conclude.

3.4 Load imbalance due to boundary conditions

The boundary conditions introduce a special source of load imbalance due to the fact that computational work to be done for boundary nodes differs from work for interior nodes. It would be no problem at all if boundary nodes will be distributed as equal as possible among the processors, but this is not always possible. In fact, for airfoils, using a mesh domain partitioning, only few processors work with the boundary nodes. The resulting impact on the parallel speedup can be investigated as follows.

We consider that there are four work units of the algorithm. Update of interiors nodes, and updates for the inlet boundary, outlet boundary and wall. Assuming that there is no communication overhead, the time to complete one iteration of the algorithm using a mesh of $p=p_x \times p_y$ processors is

$$T_p = \max_{j=1,p} \{f_{int} + f_{inl} + f_{out} + f_{wall}\} t_t \tag{11}$$

The number of floating point operations f_i can be assumed to be analogous to number of nodes participating in the work unit

$$f_i = C_i \frac{N_i}{P_i} \tag{12}$$

Assuming ,without loss of generality ,that $C_{inl}=C_{out}=C_{wall}=(1+a)C$ (where C floating point operations per interior node), and that there is an equi-node distribution along the processors in x and y directions ,(the most overloading processors are those which contain both inlet and outlet boundary nodes),we get

$$T_p = \frac{Ct_f}{P} \left\{ N - N_b + (1+a) \left(N_x P_y + \frac{N_y P_x}{2} \right) \right\} \tag{13}$$

where $N=N_x N_y$, $N_b=N_x+N_y+N_w$
For P=1, we have to include work in wall and in all outlet, so we get

$$T_1 = Ct_f (N + aN_b) \tag{14}$$

Assuming normal values for the parameters appearing in the above relations we obtain speedup less than the theoretical (P).This proves that special care has to be given to boundary nodes work in order to minimize parallel inefficiency.

4. Parallel Impementation and Results.

The quality of the results obtained by the serial version of the code has already been verified in quite a few cases [6-7]. The developed parallel code has been tested both for its integrity and effectiveness. The chosen test case for the parallel tests was the prediction of flow around a NACA012 airfoil. A C-type mesh (161X37) was used. First the code was validated in order to be sure that the new machine independent parallel code produces results identical with the serial one. Secondly, speedup and efficiency were investigated for different machines. In the following figures results from execution of the code on three different machines (SGI Indigo WS,PARAGON ,GCel) are shown.From Figure 1 where the parallel efficiency is shown we can conclude that for achieved medium grain parallelization (up to 32 processors) the efficiency is satisfactory (more that 75%). Obviously the reduction of the efficiency as more processors are involved is due to the size

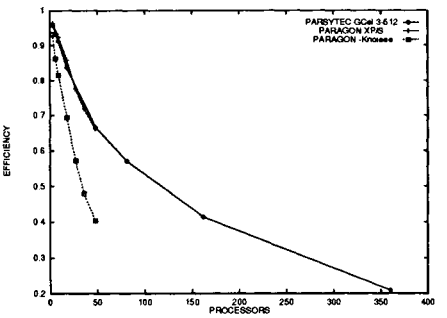


Figure 1

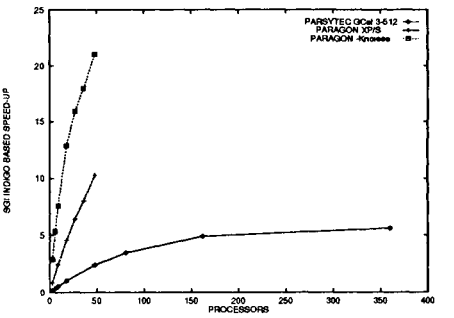


Figure 2

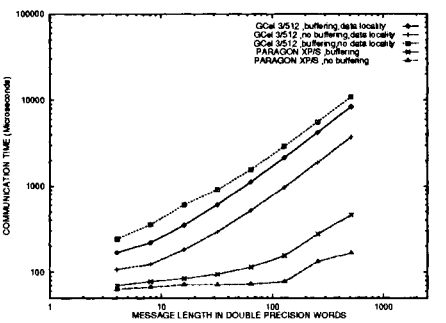


Figure 3

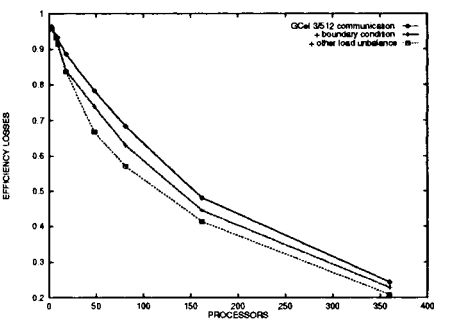


Figure 4

of the problem considered. A hard evidence that the traditional definition of parallel efficiency is not always an appropriate indicator for the actual benefit obtained through parallelization, is given in Figure 2. It can be seen that though the efficiency of the code in Paragon when using the compiler option - Knoiee is lower than the efficiency in the Parsytec machine, the effective speed-up is about 20 times higher.

In order to estimate the characteristic communication parameters required for the parallel performance model we developed in section 3.2, the Communication Library mentioned in section 3.1 has been utilized for benchmarking the used parallel platforms. The results are shown in Figure 3. Three main conclusions are drawn from the figure. First, necessary buffering when two and three indexed arrays have to be exchanged, has a considerable impact on the transfer bandwidth. Second conclusion is that though the

bandwidth in Paragon is much higher than in Parsytec, the latency is only slightly lower. The concequence is that the Paragon machine is not appropriate for fine grain parallelism in small size problems, given that its computing power is much higher than in Parsytec. The third conclusion is that data locality (efficient mapping) is crucial in GCel.In Figure 4 are illustrated the proportional losses of efficiency, obtained with the help of the parallel performance model. It can be seen that a significant portion of the efficiency loss is due to load imbalance imposed by the boundary conditions.

Finally, investigations regarding the impact of the loose parallel coupling (communication in boundaries only after certain iterations) have been carried out.The conclusion of these investigations is that there is no benefit in terms of the obtained speedup due to the considerable decrease in the convergence rate of the algorithm.

REFERENCES

1. Johnson S L,Saad V.,Schultz M.,H.,(1987), "Alternating Direction Methods on Multiprocessors" ***SIAM J. Sci. Stat. Comput.*,Vol. 8, pp. 686-700.**
2. Laval, P., 1983, "Nouveaux Schémas de Désintégration pour la Résolution des Problèmes Hyperboliques et Paraboliques Non Linéaires: Application aux Equations d' Euler et de Navier-Stokes", ***Recherche Aérospatiale*, No 4.**
3. Abarbanel, S. and Gottlieb, D., 1981, "Optimal Time Splitting for Two- and Three-Dimensional Navier-Stokes Equations with Mixed Derivatives", ***Journal of Computational Physics 41.***
4. MacCormack, R.W., 1988, "On the Development of Efficient Algorithms for Three-Dimensional Fluid Flow", Recent Developments in Computational Fluid Dynamics, T.E. Tezchigar et al., ed., **ASME AMD-Vol. 95, pp.117-138.**
5. Jameson, A., Schmidt, W. and Turkel, E., 1981, "Numerical Solutions of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time Stepping Schemes", **AIAA Paper 81-1259.**
6. Simandirakis, G., 1992, "Numerical Solution of Navier-Stokes Equations for Transonic Flows Inside Turbine Bladings", **PhD Thesis**, NTUA, Athens, February 1992.
7. Simantirakis g.,Giannakoglou K.,Alkalai k.,Papailiou K. D., "Development and Application of a Fractional Step Method for the Solution of Transonic and Supersonic Flow Problems", **Proc. of Fifth International Symposium on Numerical Methods in enginnering.**
8. Otto J. C,"Parallel Execution of A Three-Dimensional,Chemically Reacting ,Navier-Stokes Code on Distributed-Memory Machines",**AIAA-93-3307-CP**,1993.
9. Barton M.,Withers G.,"Computing performance as function of the speedup,quantity, and cost of the processors" ,**Proc. Supercomputing '89,1989,pp. 759-764.**
10. Sun X., Gustafson J.,'Toward a better parallel Performance metric',***Parallel Computing* 17, pp 1093-1109**, 1991.
11. Van Catlege F.A. "Toward a General Model for Evaluating the Relative Performance of Computer Systems",**Int. J. of Supercomputer Applications, 2, 100-108**, 1989.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallelization of a Highly Unstructured Euler-Solver Based on Arbitrary Polygonal Control Volumes

Clemens Helf, Klaus Birken, Uwe Küster *

The paper describes a strategy to obtain SPMD-parallelizations by using an abstract model for distributed, dynamic data. The abstract model encapsulates the distributed aspects of the application and provides an application-oriented interface to message passing. The strategy is applied to a self-adaptive Finite Volume scheme based on arbitrary polyhedral control volumes. Efficiency and numerical results are presented.

1. DESCRIPTION OF THE EULER SOLVER

The algorithm is a cell-centered finite volume method with higher order reconstruction [2,9,10] in two and three space dimensions with explicit integration in time.

The reconstruction is based on derivatives calculated from a locally fixed neighbourhood and ensures positivity of the relevant unknowns density, pressure and internal energy. The derivatives are limited so that the reconstruction does not exceed the extremal values of the stencil. Roe's flux difference splitting with entropy fix [7] is applied to the values reconstructed at side centers to determine flux contributions across each side of the control volume. Boundary conditions are enforced by a layer of ghost volumes which are equipped with appropriate state values.

2. GRID REPRESENTATION

A flexible grid representation for finite volume methods is proposed. Polyhedra with an arbitrary number of sides are used as control volumes. Those control volumes even may be non-convex or multiply connected. Control volumes are described by a hierarchy of geometric objects (nodes, edges, faces, cells) of different dimension and topological relations between them [6].

The chosen grid representation provides a unified view of the computational domain in two and three space dimensions. This unified view is presented as an interface to the algorithmic part of the code and allows to integrate grid generation into the simulation process.

*University of Stuttgart, Computing Center (RUS) Allmandring 30, D-70550 Stuttgart, Germany; EMail helf@rus.uni-stuttgart.de

3. GRID GENERATION AND ADAPTIVITY

Due to the flexibility of the chosen grid representation, the initial description of the computational domain by its boundaries already describes an admissible, single control volume. Hence, no initial grid must be generated and simulations may easily be started from scratch.

In order to supply meaningful space discretizations, control volumes may be divided along arbitrary intersection planes. Successive application of simple refinement steps (e.g. along coordinate lines) will provide initial grids for arbitrary complex domains without additional human effort.

In comparison to usual refinement procedures, an additional degree of freedom is available with refinement. In addition to the location of the refinement, the direction of the intersection plane may be freely chosen. For self-adaptive refinement, usual indicator functions [8] (e.g. pressure derivatives) may be used to determine location and direction of the refinement.

4. PARALLELIZATION STRATEGY

4.1. SPMD Parallelization

On a distributed-memory parallel computer architecture, the whole grid database has to be split up into overlapping subdomains, each of them mapped into one processor's local memory [5]. The management of data redundancy and consistency of the distributed, partly replicated database is a highly complex task and may easily prevent successful parallelization. Additionally, this task is complicated by the requirements of adaptive applications (dynamic redistribution, dynamic load balancing).

The distributed aspects of an application's database, introduced by domain decomposition, are usually not part of the application model (SPMD paradigm). So, these details are preferably invisible to the application and therefore may be seperated into a distinct layer. This was achieved by using the *Dynamic Distributed Data* library (DDD), developed at the Computing Center, University of Stuttgart [4].

4.2. A Model For Dynamic Distributed Data

DDD implements an abstract data model, which defines the relationship between a global data structure (object) and a distributed data structure, using an exact formalism. References provide a common technique for representing connectivity relations between objects (e.g. between triangles and nodes in an unstructured grid). Objects and references build up an abstract graph, which describes the database (see figure 1).

Admissible graph distributions are easily constructed and serve as a means to build up abstract interfaces, which describe local memory borders within the distributed database (see figure 2). Interfaces are characterized by their communication strategy and are kept consistent with all changes of the distributed graph. The *lean consistency* paradigm supports the design of consistency protocols which are governed by numerical demands [3].

4.3. Handler Interface

In order to appropriately execute object transfers and data updates, certain procedural information about the application database must be provided to the DDD layer. This information is supplied by a set of handlers (call-back functions), which need to be adapted to the application. Especially, handlers exist so that applications which are equipped with their own memory management may successfully coexist with memory operations induced by DDD.

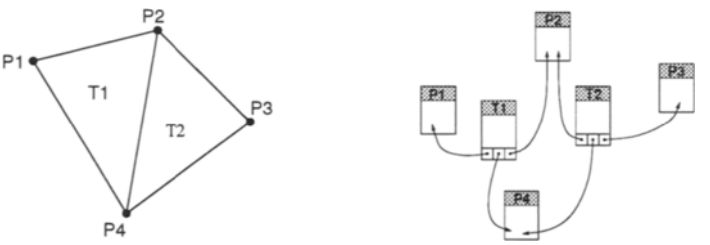


Figure 1. Detail of an unstructured grid and corresponding data model.

5. PARALLELIZATION OF THE EULER CODE

The parallelization of the code was relatively easy to achieve and first parallel jobs could be started within days. The structure of the sequential application was completely left intact. From the software engineering point of view and taking into account the complexity of the database this proves the validity of the concept. All parts of the application (I/O, simulation, refinement) were parallelized.

5.1. Introducing DDD

In a first step, a DDD data structure was inserted into the sequential application's data hierarchy, which is straightforward in programming languages supporting user defined data structures. The properties of this structure are part of the DDD paradigm, hence the complete DDD functionality may be used immediately after this step.

The following calls were inserted into the sequential code:

- a startup/shutdown call for the DDD library (main program),
- a call to the DDD transfer module to distribute a locally created grid (application startup),
- interface synchronization calls (timestep loop).

Figure 2 presents a distribution of the database from figure 1, where local copies of objects in the overlap region were introduced by DDD. The right part of figure 2 shows the interface (dotted circles), built up during data distribution. A data synchronization call causes data transfers across the interface (arrows).

5.2. Handler Interface

In a second step, handlers were introduced, which provide application specific functionality to DDD:

- memory management handlers for malloc/free-type operations,
- object management handlers for object creation and removal,
- object transfer handlers, which implement the overlap strategy,
- interface synchronization handlers, specifying the data to be transferred.

After this step, an operational parallel code was available.

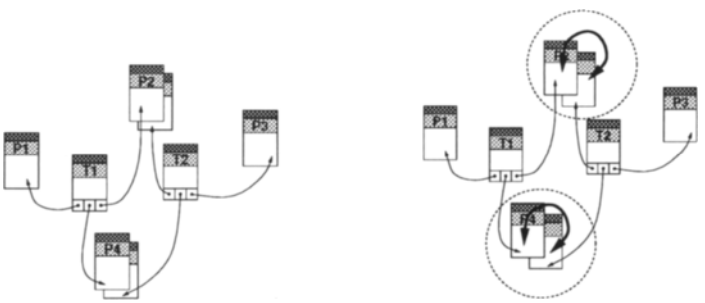


Figure 2. Distributed database with local copies.

5.3. Grid Refinement

In a third step, the refinement procedures were reviewed for parallelization. From the viewpoint of the grid database, refinement consists of substituting existing objects of the grid by new objects. In order to keep the distributed data management up to date, the object creation/removal interface of the database was split into a local object layer and a distributed object layer, which contains the corresponding DDD calls.

All communication needed in the overlapping regions of the distributed grid could be introduced by adding a few DDD transfer commands.

5.4. Conclusion

Even an extremely complex, dynamic database may easily and quickly be prepared for use in a SPMD-type parallelization. The possibility to provide application specific handlers allows to use DDD in a wide variety of applications. However, the introduction and maintenance of these handlers still requires substantial care and experience.

Nearly no modifications are necessary to the applications source code. The structure of the sequential application was completely left intact. So, further improvements in the CFD part of the code can be realized without impact on parallelization.

As DDD itself is available for a variety of architectures (Intel Paragon, Cray T3D, Workstation Cluster) respectively message passing systems (NX, PVM, MPI), portability is not an issue.

It may be concluded that the DDD paradigm perfectly fits into the framework of a dynamic application for complex databases.

6. RESULTS

6.1. Efficiency Measurements

The following figure 3 presents parallel efficiency measured on an Intel Paragon system. Measurements are obtained from a supersonic doublewedge test case [1], which adaptively develops from 256 to 1660 control volumes. Hence, higher efficiencies may be expected for larger problems.

The lower measurement includes the total parallel overhead (i.e. load balancing, dynamic re-distribution, data synchronization at processor interfaces), while the second measurement excludes data synchronization times.

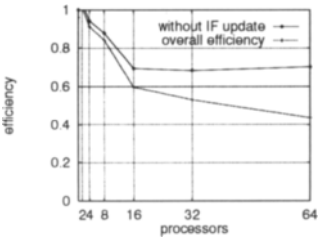


Figure 3. Parallel efficiency for a self-adaptive computation.

6.2. Numerical Example: Doublewedge

Figure 5 shows a calculation of a Mach 3 duct with a doublewedge (16.7° angle of attack) on twenty processors, cf. [1]. The grid contains 10294 control volumes. Local memory borders are shown within the solution plot.

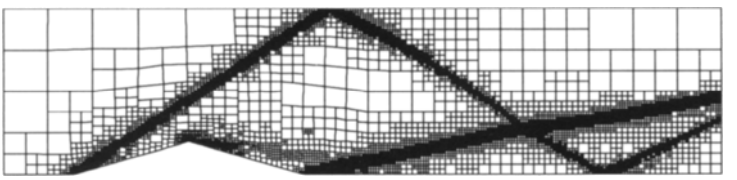


Figure 4. Grid with 10294 control volumes.

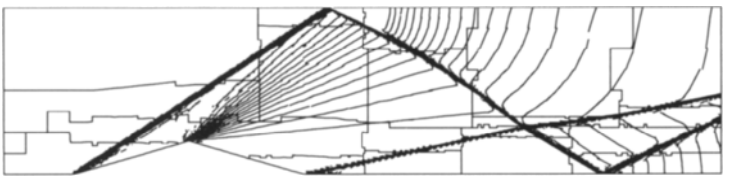


Figure 5. Pressure isolines and local memory borders.

7. OUTLOOK

The introduction of second-order reconstruction and Navier-Stokes equations into the CFD code is currently going on. The grid database will be enhanced to support multi-level grids, which will serve as a base for instationary refinement, parallel re-coarsening and fast solution techniques.

Further performance investigations will help to improve parallel efficiency. Different load balancing tools will be investigated and prepared for use with DDD. Tools for the automatic creation of handlers are under consideration.

REFERENCES

1. M.J. Aftosmis and N. Kroll. A quadrilateral based second-order tvd method for unstructured adaptive meshes. Report 91-0124, AIAA, 7-10 Jan. 1991.
2. T.J. Barth. Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes equations. In *Special Course on Unstructured Grid Methods for Advection Dominated Flows*, AGARD Report R-787, Neuilly Sur Seine, France, May 1992.
3. K. Birken. An efficient programming model for parallel and adaptive CFD-algorithms. In *Proceedings of Parallel CFD Conference 1994*, Kyoto, Japan, 1995. Elsevier Science.
4. K. Birken and P. Bastian. Dynamic Distributed Data (DDD) in a parallel programming environment - specification and functionality. Forschungs- und Entwicklungsberichte RUS-22, Rechenzentrum der Universität Stuttgart, Germany, September 1994.
5. J. DeKeyser. *Solution of the Steady Euler Equations by Means of Adaptive Unstructured Meshes, Multi-Grid Methods and Parallel Computers.* PhD thesis, Catholic University Of Leuven, Leuven, Belgium, 1994.
6. C. Helf and U. Küster. A finite volume method with arbitrary polygonal control volumes and high order reconstruction for the Euler equations. In S. Wagner, E.H. Hirschel, J. Périaux, and R. Piva, editors, *Proceedings of the Second European Computational Fluid Dynamics Conference*, Stuttgart, Germany, 1994. Wiley & Sons.
7. C. Hirsch. *Numerical Computation of Internal and External Flows - Volume 2: Computational Methods for Inviscid and Viscous Flows.* Wiley & Sons, 1990.
8. R. Löhner. Finite element methods in CFD: Grid generation, adaptivity and parallelization. In *Special Course on Unstructured Grid Methods for Advection Dominated Flows*, AGARD Report R-787, Neuilly Sur Seine, France, May 1992.
9. P. Vankeirsbilck. *Algorithmic Developments for the Solution of Hyperbolic Conservation Laws on Adaptive Unstructured Grids (Application to the Euler Equations).* PhD thesis, Van Karman Institute, University Of Brussels, Brussels, Belgium, 1993.
10. M. Wierse. *Higher Order Upwind Schemes on Unstructured Grids for the Compressible Euler Equations in Timedependent Geometries in 3D.* PhD thesis, Albert-Ludwigs-Universität, Freiburg i. Br., Germany, 1994.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Performance of a Euler Solver Using a Distributed System

Renato S. Silva and Regina C. Almeida * [a]

[a]Laboratório Nacional de Computação Científica
Rua Lauro Müller, 455, 22290-160,
Rio de Janeiro, Brazil

Supercomputers nowadays can be used to solve large-scale problems that come from simulation of industrial or research problems. However, those machines are usually inacessible to most industries and university laboratories around the world. In this work we present an Euler solver to be used in a collection of workstations under PVM that can overcome the use of direct methods on a single processor.

1. Introduction

In the last years, with the advent of successful discretization methods and powerful computer technologies, computer simulation has been seen as a valuable approach to be combined with experimental methods to solve industrial and scientific problems. In this work we are interested in the numerical simulation of compressible Euler flows for which we have to deal with large, sparse and non-symmetric systems. For these types of problems the computational cost is a major concern and so is the accuracy. Obviously, the accuracy is related with the appropriate choice of the numerical method to solve the Euler equations.

Ideally, the numerical method to solve the compressible Euler equations should have good convergence properties and should be stable in the presence of shocks: in those regions of the fluid flow, the variables in the system vary strongly, making the computation very challenging. In that case, it is well known that the approximate solution obtained by using the standard Galerkin finite element method is completely spoiled by spurious oscillations that are spread all over the computational domain. In order to avoid those oscillations, we shall use a very stable Petrov-Galerkin method called CAU Method (Consistent Approximate Upwind Method)[4], that consistently adds to the Galerkin weighting functions two terms. The first one acts over the generalized streamline direction which comes from the SUPG Method (Streamline Petrov-Galerkin Method)[1] and the second one provides the control over the derivatives in the direction of the generalized approximate gradient. This method is also very accurate since the latter perturbation vanishes in the regions where the solution is regular. However, there still be the necessity of using an efficient solver to reduce the computational costs when large problems are concerned.

*Visiting Scholars from TICAM - Texas Institute of Computational and Applied Mathematics, The University of Texas at Austin. During the course of this work the authors were supported by the Brazilian Government fellowship CNPq proc. 201387/93-0 and 200289/93-4, respectively.

It is well known that iterative methods perform better than direct methods for large problems, but to determine the best iterative algorithm to solve non-symmetric systems is still an open issue. However, many conjugate gradient like methods have been widely used with relative success. Included in this class, the GMRES method introduced by Saad and Schultz seems to be a good choice because it seldom breaks down and presents monotonic convergence. But the method possesses a drawback because, in its original version, the orthogonalization process dictates an increasing amount of computational work with increasing iterations. This drawback can be avoided using a restarted GMRES version (GMRES(k)) since it consumes less memory than the original one while keeping its good properties.

A good solver means not only a choice of the best method but a combination between the method and a fast implementation. Recently considerable attention has been focused in heterogeneous distributed systems as a solution for high performance computations. By definition, a heterogeneous distributed system is a collection of different computers loosely connected by a Local Area Network (LAN) or/and a Wide Area Network (WAN). The popularity of these systems can be explained by the increasing performance and lower prices of high speed networks and general-purpose workstations. This means that is much cheaper to form and to maintain these systems than a massively parallel machine (MPM) and, in certain cases, their performances can be similar.

Unlikely MPMs, heterogeneous distributed systems can be formed with different types of processors and with different network types. Besides, they are composed by general-purpose enviroments, which means that it is possible to have different users running their jobs in the machines of the system. As a consequence, the load of each machine and the network traffic can vary at each instant and the choice of the communication pattern has to be done carefully.

Thus, in this work, we present a distributed implementation of the GMRES(k) with right preconditionig, for which the computational cost depends on the dimension of the Krylov Space (k) and on the interface of the domains and we show that this algorithm running on a collection of 16 IBM RISC 6000 workstations can get satisfactory efficiency with a relative small number of degrees of freedom.

An outline of this paper is as follows. In section 2 the CAU method for the compressible Euler equations employing entropy variables is presented. In section 3 the GMRES method and the Additive Scharwrz method used as preconditioner are discussed. Section 4 describes our distributed implementation of methods shown on previous sections. Numerical results are presented in section 5 and the conclusions of this work are drawn in section 6.

2. Simulation of Compressible Inviscid Flows

In this work we are interested in the numerical simulation of the steady state solution of the two-dimensional compressible Euler equations. For such non-linear problems, we get the steady state solution from the limit solution of a time dependent problem. We shall use a stable Petrov-Galerkin formulation, the *CAU method*, in order to prevent instabilities when sharp gradients are presented in the solution [see [4,5] for details].

2.1. Approximate Solution - The CAU Method

We shall formulate the CAU method for the Euler equations by using the time-discontinuous Galerkin method as the basis of our formulation. To this end, consider partitions $0 = t_0 < t_1 ... < t_n < t_{n+1} < ...$ of $\Re^+$ and denote by $I_n = (t_n, t_{n+1})$ the $n^{\underline{th}}$ time interval. The space-time integration domain is the product $\Omega_n = \Omega \times I_n$, $\Omega \subset \Re^d = \Re^2$, with boundary $\overline{\Gamma} = \Gamma$ x I_n. Denote by Ω_n^e de e^{th} element in Ω_n, $e = 1, ...(N_e)_n$, where $(N_e)_n$ is the total number of elements in Ω_n. For $n = 0, 1, 2...$, let us first introduce the set of kinematically admissible functions

$$S_n^h = \left\{ V^h ; V^h \in \left(C^0 (\Omega_n) \right)^m ; V^h \mid_{\Omega_n^e} \in \left(P^k (\Omega_n^e) \right)^m ; f^1 \left(V^h \right) \mid_{\overline{\Gamma}_n} = g(t) \right\}$$

and the space of admissible variations

$$\Im_n^h = \left\{ \widehat{V}^h ; \widehat{V}^h \in \left(C^0 (\Omega_n) \right)^m ; \widehat{V}^h \mid_{\Omega_n^e} \in \left(P^k (\Omega_n^e) \right)^m ; f^2 \left(\widehat{V}^h \right) \mid_{\overline{\Gamma}_n} = 0 \right\},$$

where f^1 and f^2 are the nonlinear boundary condition transformation, g is a prescribed boundary condition and P^k is the space of polynomials of degree less or equal to k.

With these definitions, the variational formulation using the CAU method consists of:

Find $V^h \in S_n^h$ such that for $n = 0, 1, 2...$

$$\int_{\Omega_n} \widehat{V}^h \cdot \left(A_0 V_{,t} + \tilde{A} \cdot \nabla V \right) d\mathbf{x}dt \quad +$$

$$\sum_{e=1}^{(N_e)_n} \int_{\Omega_n^e} \left(A_0 \widehat{V}^h + \tilde{A} \cdot \nabla \widehat{V}^h \right) \cdot \tau_s^e \left(A_0 V_{,t} + \tilde{A} \cdot \nabla V \right) d\mathbf{x}dt \quad +$$

$$\sum_{e=1}^{(N_e)_n} \int_{\Omega_n^e} \left(\left(\tilde{A} - \tilde{\theta}^h \right) \cdot \nabla \widehat{V}^h \right) \cdot \tau_c^e \left(A_0 V^h + \tilde{A} \cdot \nabla V^h \right) d\mathbf{x}dt \quad +$$

$$\int_{\Omega} \widehat{V}^h (t_n^+) \cdot A_0 \left(V^h (t_n^+) - V^h (t_n^-) \right) dx = 0 , \qquad \forall \widehat{V}^h \in \Im_n^h . \tag{1}$$

$V = (V_j)_{j=1}^m$ is the entropy variables vector, m is the number of variables ($m = 4$ for the two-dimensional case); $A_0 = \frac{\partial U}{\partial V}$ is a (mxm) symmetric and positive-definite matrix; $U^t = [\rho, \rho u_1, \rho u_2, \rho e]$ is the conservation variables vector; ρ is the density; u_i is the velocity in $i^{\underline{th}}$ direction, $i = 1, 2$; e is the total energy density. $\tilde{A}_i = A_i A_0$ is symmetric, where $A_i = \frac{\partial F_i}{\partial U}$ is the Jacobian matrix, F_i is the Euler flux. Finally, $\nabla^t (\cdot) = \left(I_m \frac{\partial (\cdot)}{\partial x_1}, I_m \frac{\partial (\cdot)}{\partial x_2} \right)$ is the generalized gradient operator where I_m is the (mxm) identity matrix and $\tilde{\theta}^h$ is an auxiliary $(mdxm)$ matrix introduced by the CAU Method. The terms in this equation can be identified in order as the standard Galerkin term, the SUPG term, the discontinuity-capturing term provided by the CAU method and the jump term by which the information is propagated from Ω_n to Ω_{n+1}. The SUPG term provides the control over the generalized streamline direction and the (mxm) matrix of intrinsic time scale τ_s^e is defined as in [8].

Introducing the definition for $\left(\tilde{A} - \tilde{\theta}^h \right)$, the CAU term can also be written as

$$\sum_{e=1}^{(N_e)_n} \int_{\Omega_n^e} \max \left\{ 0 \quad , \quad h_c^e \frac{\left| \mathcal{L}^h \right|_{A_0^{-1}}}{\left| \nabla V^h \right|_{A_0}} - \frac{\nabla V^{h^t} A \, \tau_s^e \, \mathcal{L}^h}{\left| \nabla V^h \right|_{A_0}^2} \right\} \nabla \widehat{V}^{h^t} [A_0] \nabla V^h \, d\mathbf{x}dt, \tag{2}$$

where h_c^e is the characteristic length in the generalized approximate gradient direction.

3. Preconditioned GMRES(k)

The GMRES method, proposed by Saad and Schultz is a Conjugate Gradient like method for solving general nonsymmetric systems of equations and has gained wide acceptance in solving systems coming from Computational Fluid Dynamics (CFD) problems [8],[9],[14]. Writing the linear system of equations generated by (1) as $Ax = b$, the GMRES derives an approximate solution to x, $x = x_0 + z$, where x_0 is the initial guess and z comes from minimizing the residual $\|b - A(xo + z)\|$ over the Krylov subspace $span\left[r_0, Ar_0, A^2r_0, ..., A^kr_0\right]$, where k is the dimension of the space and $r_0 = b - Ax_0$. The major drawback of GMRES is that increasing k the amount of memory required per iteration increases linearly and the computational work increases quadratically . In order to control the amount of work and the memory requirements a restarted version (GMRES(k)) is used. This means that after k orthogonalization steps all variables will be cleaned and the obtained solution will be taken as the initial solution for the next k steps.

The crucial element for a successful use of this version is based on the choice of the parameter k. If it is too small, the convergence may slow down or the method can not converge, but with large value the work and the memory requirements would turn this method useless.

Another important point is the choice of the preconditioner. We choose the Addive Scharwz Method (ASM) [11] which not only can increase the convergence rate but has a high degree of parallelism. The main idea of the ASM is that we can solve the preconditiong step of the GMRES algorithm [13] $z_j = M^{-1}v_j$, by partioning the domain Ω in p overlaping subdomains Ω_i and then approximate A^{-1} by $M^{-1} = \sum_{i=1}^{P} R_i^t \; A_i^{-1} \; R_i$ where A_i is the local submatrice formed by $A_i = R_i \; A \; R_i^t$ with R_i and R_i^t the restriction and extension matrices. It can be noticed that the preconditioning step can be solved in parallel with each processor solving a smallest system of equations. There are two forms to solve the local problems: a direct solver or a iterative solver, like multigrid. At this time we decided to use a direct solve, the LU decomposition method from the Lapack package[16].

4. Distributed Implementation

Recently there has been much interest in using workstations to form distributed systems. With the quickly evolution of desktop workstations, they can off-load jobs from saturated vector computers, often providing comparable term around time at a fraction of the cost. With high-speed networks the workstations can also serve as an inexpensive parallel computer. Another factor in favor of distributed compunting is the availability of many lightly loaded workstations. These otherwise wasted idle cycles can be used by a distributed computation to provide speedups and/or to solve large problems that otherwise could not be tackled.

Similar to other Conjugate Gradient like iterative methods, the most expensive part of the GMRES algorithm is the preconditioner. Using the ASM the preconditioner step is reduced to a series of local sequential problems. But now the question is: *how to*

partition the data among the processors? The significant fact for this type of system is that the common network used, Ethernet and FDDI, allows all processors to logically communicate directly with any other processor but physically supports only one '*send*' at a time. So we chose one-dimensional partition of the domain and we associate each one with a different processor, reducing the overhead on '*startups*' in the network and the number of neighbors. This type of partition will lead the way that the matrix-vector products will be done.

With this distribution each processor will have a part of the initial residual vector r_0 and part of the matrix-vector result, the v_i vectors. Thus, the orthogonalization will be performed in parallel, where the partial results ($\|r_0\|$,$\|v_i\|$, $\beta_{i+1,j}$) will be exchange to complete the process.

Obviously this procedure requires a greater number of communications. That is increased by the fact that an Element-by-Element structure is being used to storage the matrix which implies that to evaluate this operation we need to do some communications between nearest-neighbors to obtain the contribution of the interface degrees of freedom. However this choice will be better to solve large problems with adaptive procedures because only fixed number of scalar variables, equal to the Krylov dimension, will be exchanged.

Two types of communication are used on this implementation: the nearest-neighbor and one global communication. For the first one we use simple '*sends*' and '*receives*' between neighbors. In the global communications we will use the Bucket Collect algorithms [15].

Recently several tools have been developed to support distributed computing like PVM, MPI and others [12]. The PVM, *Parallel Virtual Machine* is chosen because it can connect a heterogeneous group of workstations without difficulty.

5. Numerical Results

In this section we convey some numerical results obtained by applying the proposed methodology on the solution of a variety of compressible Euler flow problems. All computations were done using piecewise linear elements in space and constant in time. This leads to a time-marching scheme which has low order of accuracy in time but has good stabilitiy properties, being appropriate for solving steady problems.

5.1. Oblique Shock Problem

This problem deals with an inviscid supersonic flow (Mach number M=2) over a wedge at an angle with respect to the mesh. It is illustrated in Figure 1 where the dashed line shows the position of the shock determined analytically.

The Figure 2 which contains the elapsed time of the solver for differente problem sizes is obtained using a collection of sixteen IBM RISC 6000 workstations model 3BT connected by FDDI network. Obviously for small degrees of freedom (dof) the performance is not satisfactory for small numbers of processors. But when the dof increases the performance increases too.

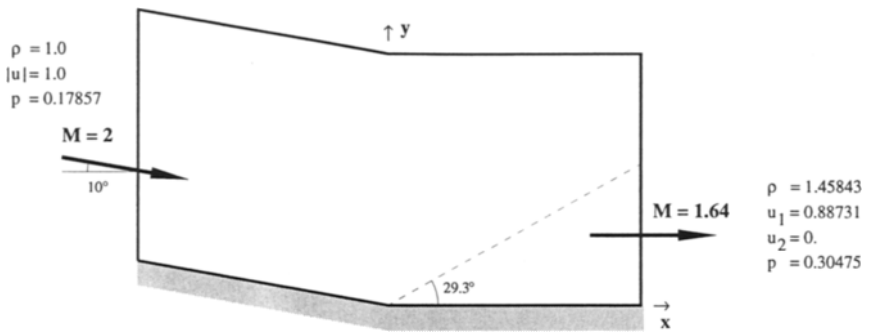


Figure 1. Oblique Shock Problem: problem statement.

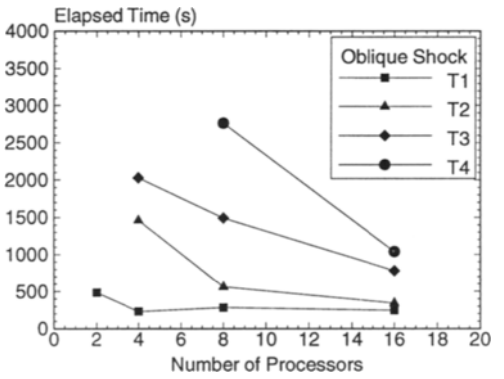


Figure 2. Oblique Shock Problem: elapsed time (s), T1 = 2,244 dof, T2 = 4,420 dof,T3 = 6,596 dof, T4 = 8,580 dof.

In Table 1 we compare the new solver with a widely used direct solver , the frontal solver, runing on a IBM RISC model 590. It can be seen that the solver introduced here in this paper is 1.8 times faster. It is important to remember that 8,580 dofs is still considered a small problem for iterative methods.

8,580 dof	
Serial (590)	Distributed (p = 16)
Frontal	ASM - GMRES(k)
31.3 min	17.3 min

Table 1. Comparison of Direct and Distributed Solver

5.2. Shock-Reflection Problem

This two dimensional steady problem consists of three flow regions separated by an oblique shock and its reflection from a wall.

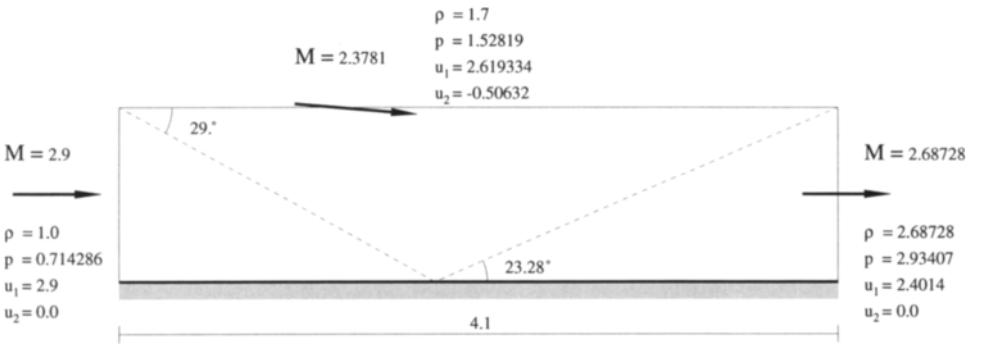


Figure 3. Shock-Reflextion: problem statement.

The timming results for this problem are very similar to the previous example. In Table 2 the comparation with the frontal solver is shown, and again our algorithm is faster by a factor 1.7.

12,740 dof	
Serial (590)	Distributed (p = 16)
Frontal	ASM - GMRES(k)
68.3 min	38.4 min

Table 2. Comparison of Direct and Distributed Solver

6. Conclusions

In this paper we have shown a distributed solver for solving CFD problems, in particular the compressible Euler equations using a stable Petrov Galerkin method written in entropy variables, designed using a space-time finite element formulation. The numerical results showed that this solver is appropriate for distributed systems like a collection of workstations connected by a token-ring or bus based network. We have shown that this type of aproach can be efficient and it can be superior or faster than the usual solvers even for the small size problems solved in this paper.

REFERENCES

1. A. N. Brooks and T. J. R. Hughes, Streamline Upwind Petrov-Galerkin Formulations for Convection-Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equations, *Comput. Methods Appl. Mech. Engrg.* 32 (1982) 199–259.
2. T. J. R. Hughes and M. Mallet, A New Finite Element Formulation for Computational Fluid Dynamics: III. The Generalized Streamline Operator for Multidimensional Advective-Diffusive Systems, *Comput. Methods Appl. Mech. Engrg.* 58 (1986) 305–328.

3. C. Johnson, U. Nävert and J. Pitkaranta, Finite Element Methods for Linear Hyperbolic Problems, *Comput. Methods Appl. Mech. Engrg.* 45 (1984) 285–312.
4. R. C. Almeida and A. C. Galeão, The Generalized CAU Operator for the Compressible Euler and Navier-Stokes Equations, *8th International Conference on Numerical Numerical Methods in Laminar and Turbulent Flows* (1993).
5. R. C. Almeida and A. C. Galeão, An Adaptive Petrov-Galerkin Formulation for the Compressible Euler and Navier-Stokes Equations, accepted for publication in *Comput. Methods Appl. Mech. Engrg.*
6. A. Harten, On the Symmetric Form of Systems of Conservation Laws with Entropy, *J. Comp. Physics* 49 (1983) 151–164.
7. T. J. R. Hughes, L. P. Franca and M. Mallet, A New Finite Element Formulation for Computational Fluid Dynamics: I. Symmetric Forms of the Compressible Euler and Navier-Stokes Equations and the Second Law of Thermodynamics, *Comput. Methods Appl. Mech. Engrg.* 54 (1986) 223–234.
8. F. Shakib, Finite Element Analysis of the Compressible Euler and Navier-Stokes Equations, *Ph. D. Thesis, Stanford University* (1988).
9. M. Mallet, A Finite Element Method for Computational Fluid Dynamics, *Ph. D. Thesis, Stanford University* (1985).
10. R. C. Almeida and R. Silva, A Stable Petrov-Galerkin Method for Convection-Dominated Problems, (submitted to *Comput. Methods Appl. Mech. Engrg.*)
11. Patrick Le Tallec, Domain Decomposition Methods in Computational Mechanics, *Computational Mechanics Advances* 1 (1994).
12. Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R. and Sunderam V., PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing, *The MIT Press*
13. Youcef Saad, A Flexible Inner-Outer Preconditioner GMRES Algorithm, *iUniversity of Minnesota Supercomputer Institute Research Report* 91/279 (1991)
14. William D. Gropp and David E. Keyes, Domain Decomposition Methods in Computational Fluid Dynamics, *Inter. Journal for Numerical Methods in Fluids* 14 (1992) 147–165
15. M. Barnett, R. Littlefield, D. G. Payne and Robert A. van de Geijn, On the Efficiency of Global Combime Algorithms for 2-D Meshes With Wormhole Routing, *TR-93-05, Department of Computer Sciences, University of Texas* 1993
16. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammariling, A. McKenney, S. Ostrouchov and D. Sorensen, *LAPACK Users' Guide, Second Edition*, SIAN, 1995

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

IMPLEMENTATION AND RESULTS OF A TIME ACCURATE FINITE-VOLUME EULER CODE IN THE NWT PARALLEL COMPUTER

L. P. Ruiz-Calavera[a] and N. Hirose[b*]

[a]INTA, Aerodynamics Division, Carretera de Ajalvir Km. 4.5, SPAIN

[b]NAL, Computational Science Division, 7-44-1 Jindaiji-Higashi,
Chofu-shi, Tokyo 182, JAPAN

A time-accurate Euler code to calculate unsteady transonic flow about wings has been implemented in the NWT parallel computer. Details of the parallel model employed together with some representative results are presented, together with information on the computer performance achieved.

1. INTRODUCTION

Aeroelastic problems appear to be of increasing importance in the design of aircraft. The size of the structures and its elastic behavior, the aerodynamic interference of different components, transonic effects, structural and control nonlinearities, are becoming a severe limiting factor, specially for the future generation of very large civil aircraft. Thus there is a strong need to apply highly sophisticated reliable aeroelastic simulation tools already in the early design stage of a new development. To clear a configuration of aeroelastic problems, a very large number of cases have to be run, performing parametric variations of: Mach number, reduced frequency, elastic modes, dynamic pressure, etc. Sophisticated time accurate CFD codes are generally considered to be too time and memory consuming for industrial application. Potential theory is mainly used whereas the next the step of approximation, i.e. Euler Equations with or without boundary layer coupling is only now slowly starting to find its way in the design offices despite the better approximation they provide. The application of high performance parallel computers to reduce the computation time is obviously extremely interesting for this kind of applications.

The objective of the present work has been to investigate the benefits of parallelizing an existing time-accurate Euler code used to calculate inviscid transonic flow around oscillating 3-D wings. The philosophy has been to minimize the number of changes to the program because of the parallelization so as to reduce the probability of introducing errors and to minimize the time needed for the implementation, rather than the attainment of extremely good parallel performance. This objective was achieved as the whole procedure took less than 2 weeks. In the following a brief description of the scheme and its parallel implementation, together with some representative results is presented.

*This work was supported by a fellowship from the SCIENCE AND TECHNOLOGY AGENCY of Japan

2. GOVERNING EQUATIONS AND DISCRETIZATION

Only a general description of the scheme is given, more details can be found in [1]. The flow is assumed to be governed by the three-dimensional time-dependent Euler equations, which for a moving domain Ω with boundary Σ may be written in integral form as:

$$\frac{\partial}{\partial t}\left(\int\int\int_{\Omega} U \, d\Omega\right) + \int\int_{\Sigma}\left[(F,G,H) - \vec{V}_{\Sigma}\cdot U\right]\cdot\vec{n}\, dS = 0 \tag{1}$$

where U is the vector of conservative flow variables, (F, G, H) are the three components of the Euler Flux vector, $\vec{V}_{\Sigma}$ is the velocity of the moving boundary, and $\vec{n}$ is the unit exterior normal vector to the domain.

This equations are solved by means of the well known finite volume algorithm formulated by Jameson, Schmidt and Turkel [2]. The domain around the wing is divided into an O-H mesh of hexahedral cells, for which the body-fitted curvilinear coordinates ξ, η, ζ respectively wrap around the wing profile (clockwise), normal and away from it, and along the span. Individual cells are denoted by the subscripts i,j,k respectively corresponding to each of the axis in the transformed plane ξ, η, ζ.

The integral equation (1) is applied separately to each cell. Assuming that the independent variables are known at the center of each cell, and taking the flux vector to be the average of the values in the cells on either side of the face and the mesh velocities as the average of the velocities of the four nodes defining the corresponding face the following system of ordinary differential equations (one per cell) results:

$$\frac{d}{dt}\left(\Omega_{i,j,k}U_{i,j,k}\right) + \left(Q_{i,j,k} - D_{i,j,k}\right) = 0 \tag{2}$$

The convective operator $Q_{i,j,k}$ constructed in this manner is second order accurate if the mesh is sufficiently smooth. This formulation is inherently non-dissipative (ignoring the effect of numerical boundary conditions), so that artificial viscosity has to be added. The well known model by Jameson with blending of 2^{nd} and 4^{th} order dissipation terms [3] is used. The idea of this adaptive scheme is to add 4^{th} order viscous terms throughout the domain to provide a base level of dissipation sufficient to prevent non-linear instabilities, but not sufficient to prevent oscillations in the neighborhood of shock waves. In order to capture shock waves additional 2^{nd} order viscosity terms are added locally by a sensor designed to detect discontinuities in pressure. To avoid overshoots near the shock waves produced by the combined presence of the 2^{nd} and 4^{th} order terms, the latter are cut off in that area by an appropriate switch. To preserve conservation form, the dissipative terms $D_{i,j,k}$ are added in the form of (1^{st} and 3^{rd} order) dissipative fluxes across each cell face.

The system of ODEs in (2) is solved by means of an explicit 5 stage Runge-Kutta scheme with two evaluations of the dissipation terms, which is second order accurate in time and can be shown [1] to have good diffusion and dispersion errors characteristics and less computational cost per time step than other schemes with a lesser number of stages .

3. IMPLICIT RESIDUAL AVERAGING

To maintain stability the explicit time-integration scheme of the preceding section has a time-step limit $(\Delta t)_{max}$ that is controlled by the size of the smallest cell. Even though

the CFL number of the 5-stage Runge-Kutta scheme is of the order of 4, the resulting Δt is usually too small for practical applications. This restriction can be relaxed by using a technique of residual averaging [4] which gives an implicit character to the time-integration scheme.

Before each time-step the residuals $R_{i,j,k} = Q_{i,j,k} - D_{i,j,k}$ are replaced by modified residuals $R^*_{i,j,k}$ which are calculated by means of an ADI method:

$$\left(1 - \epsilon_{i,j,k}\delta^2_\xi\right)\left(1 - \epsilon_{i,j,k}\delta^2_\eta\right)\left(1 - \epsilon_{i,j,k}\delta^2_\zeta\right) R^*_{i,j,k} = R_{i,j,k} \tag{3}$$

where δ^2_ξ, δ^2_η, and δ^2_ζ are the second difference operators in the ξ, η, and ζ directions and $\epsilon_{i,j,k}$ is the smoothing parameter [5]

$$\epsilon_{i,j,k} = max\left\{\frac{1}{4}\left[\left(\frac{\Delta t}{\Delta t_{max_{i,j,k}}}\right)^2 - 1\right], 0\right\} \tag{4}$$

with Δt denoting the desired time step, and $\Delta t_{max_{i,j,k}}$ the locally maximum allowable time step.

Within a linear analysis, the former technique assures unconditional stability for any size of the time step. However, as the resulting effective Courant number becomes large the contribution of the dissipation terms to the Fourier symbol goes to zero, and consequently, the high frequencies introduced by the non-linearities are undamped [6]. Thus the practical limit for the time step is determined principally by the high frequency damping characteristics of the integration scheme used. As the properties of the 5-stage Runge-Kutta time-integration method are very good from this point of view, CFL values as high as 240 have been successfully used, which significantly decrease the calculation time needed for a typical case.

4. NWT COMPUTER

The above presented scheme was originally developed in a Cray-YMP computer and has been implemented in the NWT (Numerical Wind Tunnel) parallel supercomputer of the National Aerospace Laboratory [7,8]. This is a distributed memory parallel machine with 140 vector processing elements (PE) and two Control Processors (CP) connected by a cross-bar network.

Each PE is itself a vector supercomputer similar to Fujitsu VP400 and includes 256 Mbytes of main memory, a vector unit, a scalar unit and a data mover which communicates with other PE's. It's peak performance is of 1.7 GFlops making it 50% faster than the standard VP400. The CPs, each with a memory of 128 MBytes, manage NWT and communicate with the front processor VP2600. The cross-bar network achieves 421 MByte/s x 2 x 142 between each processor. The resulting total performance of NWT is 236 GFlops and 35 GBytes.

The specific feature of NWT is that its architecture is selected with the sole intention of obtaining maximum efficiency when running CFD applications. Each PE itself can execute a large scale computation with few data exchange with other PE's.

5. PARALLEL COMPUTATION MODEL

The code has been parallelized using Fujitsu NWT FORTRAN which is a FORTRAN 77 extension to perform efficiently on distributed memory type parallel computers. The extension is realized by compiler directives. Basic execution method is the spread/barrier method.

The present scheme has always two directions in which the computation can be performed simultaneously. Accordingly we can use one direction for vectorization and the other for parallelization. For the O-H grid used here the most natural way of parallelizing, i.e. assigning different vertical grid planes to different processing elements has been used. We thus divide every array evenly along the k-index and assign each part to different PEs. The vectorization is made in i-direction which usually has the largest number of cells.

With this partition, i-derivatives and j-derivatives can be computed in each PE without any communication. The computation of k-derivatives in PE_k requires data stored in PE_{k+1} and PE_{k-1} which, in principle, would imply the need to communicate with the neighbor PEs, thus increasing the overhead. This can be partly avoided using the concept of overlapped partitioned arrays. This is an extremely efficient concept which allows us to specify an array partition so that adjacent partitioned ranges automatically overlap and have some common indices, that is, copies of selected data at the interfaces between two PEs are stored at both local memories. In this way k-derivatives can also be computed in each PE without any communication. At the end of each calculation cycle, data in the overlap range of the partitioned arrays is harmonized by copying its value from the parent PE.

The above explained procedure can be maintained throughout the code except at the residual averaging subroutine, where the alternating directions method (ADI) employed prevents its use as it requires a sequential calculation. The inversions in the i- and j-directions can be done in each PE independently so that the k- parallelization can be maintained, with the vectorization in j-direction for the i- inversion and in i-direction for the j-inversion. As for the k-inversion, the process must be sequential in the k-direction so that we transfer the affected data from a k-partition to a j-partition. This is made asynchronously. Then we can compute the k-inversion on each PE with vectorization in i-direction. At the end of the calculation the data is transferred back to a k-partition. Figure 1 depicts the calculation flow.

6. RESULTS

Calculations have been performed for the LANN wing. This is a high aspect ratio (7.92) transport-type wing with a 25° quarter-chord sweep angle, a taper ratio of 0.4, and a variable 12% supercritical airfoil section twisted from about 2.6° at the root to about −2.0° at the tip. The geometry used for the computational model is that of [9]. The results presented here correspond to the design cruise condition: M =0.82, $\alpha = 0.6°$. The wing performs harmonical pitching oscillations about an axis at 62% root chord with an amplitude of $\alpha_1 = 0.25°$ and a reduced frequency k=0.104.

The calculation proceeds as follows: first an initial steady solution is obtained and quality controlled; then the time-accurate calculation is started and is time-marched until the initial transitories are damped and an harmonic solution is obtained (typically three

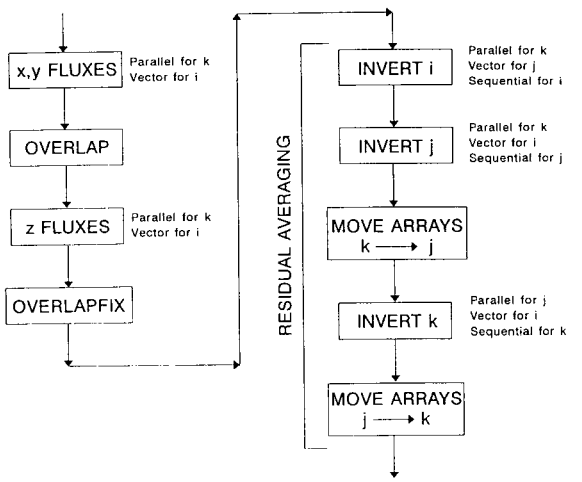


Figure 1. Parallelization-Vectorization strategy

cycles of oscillation are needed); finally the results of the last cycle are Fourier analyzed to extract the mean value and harmonics of the different aerodynamic coefficients. The instantaneous grid is computed through deformation of the steady grid system in such a way that the grid points near the wing surface are forced to closely follow the motion of the wing whereas the displacements of grid points far from the wing surface gradually decrease and vanish at the outer boundary, which remains stationary thus facilitating the implementation of the boundary conditions.

Because of the large memory and CPU time requirements of this type of methods, very few studies are available in the literature that assess the relative influence on the unsteady results of the different parameters that control the calculation. To take advantage of the benefits of parallelization to perform this task was one of the main objectives of the present work. Many different cases and parameters have been considered. Because of space limitations only two of them will be presented here.

6.1. Grid Size

Two different grids, namely 80x16x30 and 160x32x30, have been considered. The smaller grid is typically used in engineering applications of this kind of methods. Results are shown in Figures 2, where the real and imaginary part of the first harmonic of the pressure coefficients around a wing section at 17.5% semi-span are presented. It can be seen that the influence is very dramatic corresponding to the better shock resolution of the finer grid.

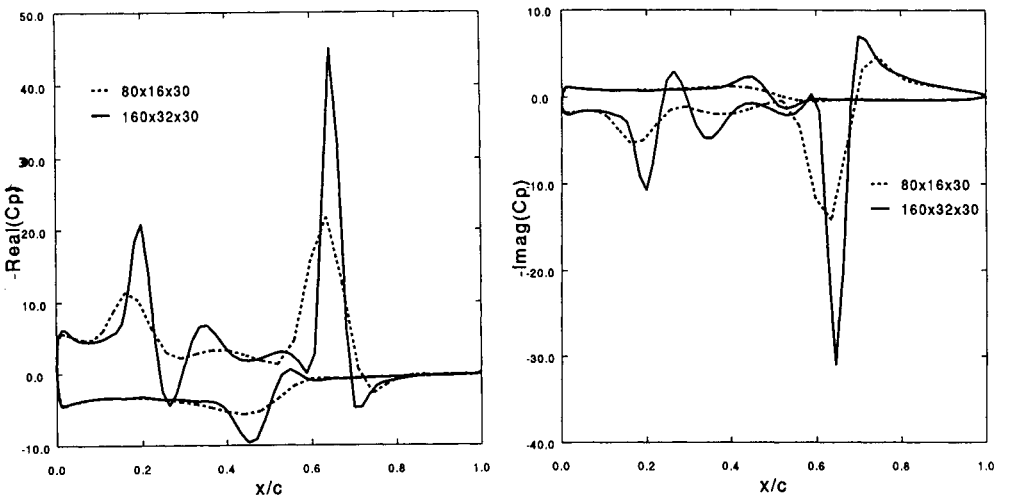


Figure 2. Effect of Grid size on First Harmonic of Pressure Coefficients around LANN wing. 17.5% semi-span

6.2. Artificial Viscosity

Calculations have been done for the 80x16x30 grid with different amounts of artificial viscosity. Results are shown in Figures 3. Logically the main effect is on the shock resolution which in turn influences the magnitude and positions of the corresponding peaks in the first harmonic component.

7. COMPUTER PERFORMANCE

In Figure 4 the speed-up factor (ratio of CPU time in 1 PE to CPU time in n PEs) vs. number of PEs used is presented for calculations performed for the LANN wing with a 160x32x30 grid. The result strongly depend on whether the residual averaging technique is used or not, because of the need to transfer data between partitions. Its relative importance in relation to the normal data transfer workload decreases as the number of PEs used increases and both curves tend to reach a common limit. It must be born in mind that the 160x32x30 grid only fills about 20% of the main memory of a single processing element (less than 1% when 32 are used), so that the granularity of the problem is extremely low. The parallel efficiency is expected to dramatically increase for larger grids, as has been the case with other codes [10].

An indication of the CPU times required to march in time the solution for the LANN wing case for one period of oscillation (using a CFL of 150 for the coarse grid and 240 for the fine one) is given in table 1:

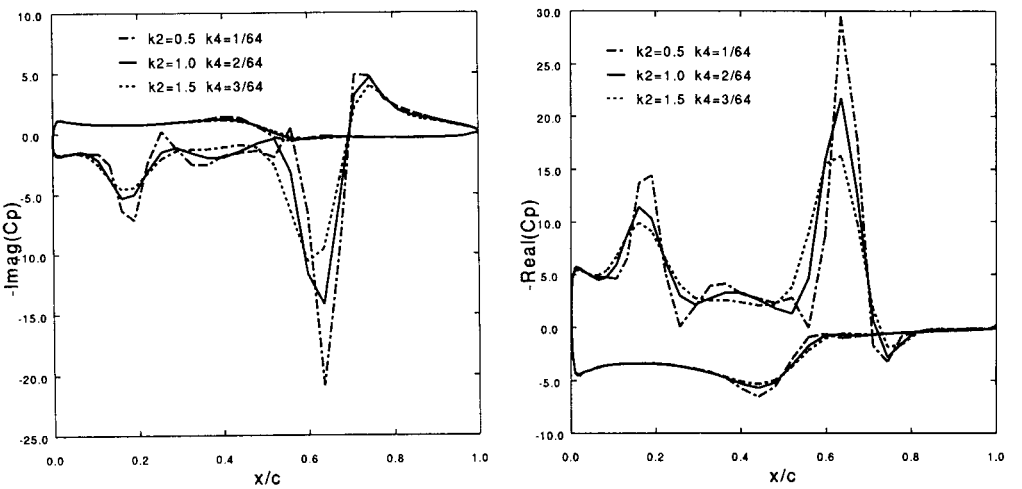


Figure 3. Effect of artificial viscosity on First Harmonic of Pressure Coefficients around LANN wing. 17.5% semi-span

Table 1
CPU Times

	NWT 1 PE	NWT 32 PE	CRAY-YMP (M92) 1 PE
80x16x30	15′	2.5′	59′
160x32x30	187′	31′	-

8. CONCLUSIONS

A time-accurate Euler code to calculate unsteady transonic flow about wings has been executed in the NWT supercomputer. The machine has proved to be extremely user friendly as the total procedure has taken less than two weeks. The achieved performance shows the feasibility of using this type of computationally expensive methods in an engineering environment.

REFERENCES

1. Ruiz-Calavera, L.P.; Hirose, N.; "Calculation of Unsteady Transonic Aerodynamic Loads on Wings Using the Euler Equations"; NAL-TR (To be published)
2. Jameson, A.; Schmidt, W.; Turkel, E.; "Numerical Solution of the Euler Equations by Finite Volume Methods using Runge-Kutta Time Stepping Schemes"; AIAA Paper 81-1259 1981

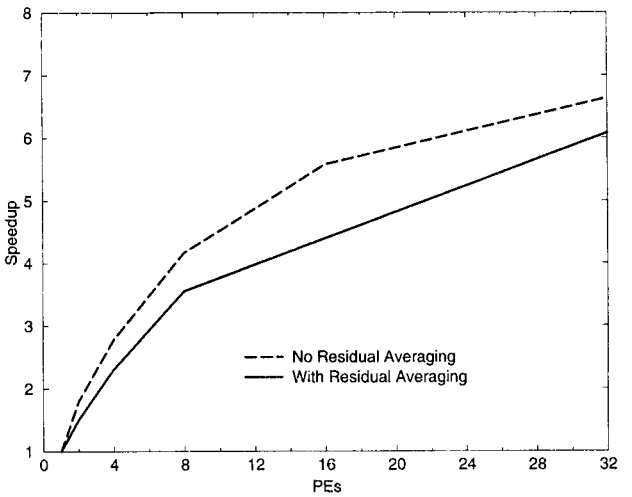


Figure 4. Speedup factor for LANN wing with 160x32x30 mesh

3. Jameson, A.; "A non-oscillatory Shock Capturing Scheme using Flux Limited Dissipation". Princeton University MAE Report 1653, 1984
4. Jameson, A.; "Transonic Flow Calculations for Aircraft"; Lecture Notes in Mathematics, Vol. 1127; Numerical Methods in Fluid Dynamics; Editor: F. Brezzi; Springer Verlag; pp. 156-242; 1985
5. Batina, J.; "Unsteady Euler Airfoil Solutions Using Unstructured Dynamic Meshes"; AIAA J., Vol. 28, No. 8, pp.1381-1388; 1990
6. Radespiel, R.; Rossow, C.; "An Efficient Cell-Vertex Multigrid Scheme for the Three-Dimensional Navier-Stokes Equations"; AIAA Paper 89-1953; 1989
7. Hirose, N.; "Numerical Wind Tunnel Project and Computational Fluid Dynamics at National Aerospace Laboratory, Japan"; NAL TM-648T
8. Iwamiya, T.; Fukuda, M.; Nakamura, T.; Yoshida, M.; "On the Numerical Wind Tunnel (NWT) Program"; Proceedings Parallel CFD '93.
9. "Compendium of Unsteady Aerodynamic Measurements"; AGARD-R-702, 1982
10. Miyoshi, H. et al.; "Development and Achievement of NAL Numerical Wind Tunnel (NWT) for CFD Computations"; Proceedings of IEEE Super Computing 1994

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Development and implementation of parallel high resolution schemes in 3D flows over bluff bodies

D. Drikakis*

Department of Fluid Mechanics, University of Erlangen-Nuremberg
Cauerstr. 4, D-91058 Erlangen, Germany

This paper presents a parallel upwind method for three-dimensional laminar and turbulent incompressible flows. The artificial compressibility formulation is used for coupling the continuity with the momentum equations. The discretization of the inviscid terms of the Navier-Stokes equations is obtained by a characteristic based method. Third order interpolation is used for the computation of the primitive variables at the cell faces. The time integration is obtained by an explicit Runge-Kutta scheme. The algorithm is parallellized using shared memory and message passing models. The latter is based either on local instructions or on PVM. The method is implemented to laminar and turbulent flows over a sphere. For high Reynolds number computations the algebraic Baldwin-Lomax as well as the $k-\epsilon$ model have been employed.

1. INTRODUCTION

Parallel computing is becoming increasingly important in the computation of complex flow problems. This is due to the demands, in terms of memory and computing time, for studying three-dimensional flow fields. The performance of a parallel computational fluid dynamics (CFD) code depends on several parameters such as: numerical methods, hardware characteristics and parallelization strategy. Widely used methods for an incompressible fluid are those based on the pressure correction procedure [1] and on the artificial compressibility formulation [2]. Explicit and implicit algorithms, on either block structured or unstructured grids, can also be combined with the above two methods.

For large scale computations single processor vector supercomputers, as well as, parallel computers have to be used. In this respect, parallel computing seems to offer the best prospect. The parallelization strategy mainly depends on the computer architecture. The most well known procedures for parallelizing an algorithm are those based on message-passing and shared memory, respectively.

The objectives of this paper are to investigate some of the above parameters using a three-dimensional incompressible Navier-Stokes algorithm and different parallelization models. The investigations are performed for three-dimensional laminar and turbulent flows over a sphere.

*current address: Department of Mechanical Engineering, University of Manchester Institute of Science and Technology, PO Box 88, M60 1QD, United Kingdom. The financial support from the Bavarian Science Foundation and the Bavarian Ministry of Education is greatly appreciated.

2. NUMERICAL METHOD

2.1. Governing equations

The governing equations are the three-dimensional incompressible Navier-Stokes equations and two additional equations for the $k-\epsilon$ turbulence model. The system of equations in matrix form and curvilinear coordinates can be written as:

$$(JU)_t + (E_I)_\xi + (F_I)_\eta + (G_I)_\zeta = (E_V)_\xi + (F_V)_\eta + (G_V)_\zeta + S \tag{1}$$

The unknown solution vector is:

$$U = (p/\beta, u, v, w, k, \epsilon)^T$$

where p is the pressure and u, v, w are the velocity components. The parameter β is the artificial compressibility parameter, and it has to be chosen to ensure the fastest convergence to steady state. The inviscid fluxes E_I, F_I, G_I, and the viscous fluxes, E_V, F_V, G_V, are written as:

$$E_I = J(\tilde{E}_I\xi_x + \tilde{F}_I\xi_y + \tilde{G}_I\xi_z) \tag{2}$$
$$F_I = J(\tilde{E}_I\eta_x + \tilde{F}_I\eta_y + \tilde{G}_I\eta_z) \tag{3}$$
$$G_I = J(\tilde{E}_I\zeta_x + \tilde{F}_I\zeta_y + \tilde{G}_I\zeta_z) \tag{4}$$

$$E_V = J(\tilde{E}_V\xi_x + \tilde{F}_V\xi_y + \tilde{G}_V\xi_z) \tag{5}$$
$$F_V = J(\tilde{E}_V\eta_x + \tilde{F}_V\eta_y + \tilde{G}_V\eta_z) \tag{6}$$
$$G_V = J(\tilde{E}_V\zeta_x + \tilde{F}_V\zeta_y + \tilde{G}_V\zeta_z) \tag{7}$$

where the fluxes $\tilde{E}_I, \tilde{F}_I, \tilde{G}_I, \tilde{E}_V, \tilde{F}_V$ and $\tilde{G}_V$ represent the corresponding Cartesian fluxes:

$$\tilde{E}_I = \begin{pmatrix} u \\ u^2+p \\ uv \\ uw \\ uk \\ u\epsilon \end{pmatrix}, \quad \tilde{F}_I = \begin{pmatrix} v \\ uv \\ v^2+p \\ vw \\ vk \\ v\epsilon \end{pmatrix}, \quad \tilde{G}_I = \begin{pmatrix} w \\ uw \\ vw \\ w^2+p \\ wk \\ w\epsilon \end{pmatrix} \tag{8}$$

$$\tilde{E}_V = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ (\mu+\mu_t/\sigma_k)\frac{\partial k}{\partial x} \\ (\mu+\mu_t/\sigma_\epsilon)\frac{\partial \epsilon}{\partial x} \end{pmatrix}, \quad \tilde{F}_V = \begin{pmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ (\mu+\mu_t/\sigma_k)\frac{\partial k}{\partial y} \\ (\mu+\mu_t/\sigma_\epsilon)\frac{\partial \epsilon}{\partial y} \end{pmatrix}, \quad \tilde{G}_V = \begin{pmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ (\mu+\mu_t/\sigma_k)\frac{\partial k}{\partial z} \\ (\mu+\mu_t/\sigma_\epsilon)\frac{\partial \epsilon}{\partial z} \end{pmatrix} \tag{9}$$

Body-fitted arbitrary coordinates are used and J is the Jacobian of the transformation from Cartesian x, y, z to generalized coordinates ξ, η, ζ. The indices ξ, η, ζ, x, y and z denote partial derivatives except for the stresses τ_{ij}. The source terms S is defined as:

$$S = (0, 0, 0, 0, P-\epsilon, f_1 c_{1\epsilon}\frac{\epsilon}{k}P - f_2 c_{2\epsilon}\frac{\epsilon^2}{k})^T \tag{10}$$

where P and μ_t are the production rate of turbulence energy and the turbulent dynamic viscosity, respectively. According to the $k-\epsilon$ model the turbulent kinematic viscosity ν_t is defined as:

$$\nu_t = f_\mu c_\mu \frac{k^2}{\epsilon} \tag{11}$$

The coefficients $f_1, f_\mu, c_{1\epsilon}, c_{2\epsilon}, \sigma_k, \sigma_\epsilon$, and c_μ can be found in Reference 3. Furthermore, the algebraic Baldwin-Lomax model [4] has also been employed.

2.2. Characteristic based method

A characteristic based method [5] is used for the discretization of the inviscid terms. The method has a similar concept with the Riemann solvers used for solving the equations of compressible flows. The characteristic based method constructs Riemann solutions on each flow direction and the primitive variables (p, u, v, w) are finally defined as functions of their values (p_l, u_l, v_l, and w_l) on the characteristics denoted by the subscript l ($l = 0, 1, 2$):

$$u = R\tilde{x} + u_0(\tilde{y}^2 + \tilde{z}^2) - v_0\tilde{x}\tilde{y} - w_0\tilde{x}\tilde{z} \tag{12}$$

$$v = R\tilde{y} + v_0(\tilde{x}^2 + \tilde{z}^2) - w_0\tilde{z}\tilde{y} - u_0\tilde{x}\tilde{y} \tag{13}$$

$$w = R\tilde{z} + w_0(\tilde{y}^2 + \tilde{x}^2) - v_0\tilde{z}\tilde{y} - u_0\tilde{x}\tilde{z} \tag{14}$$

where

$$R = \frac{1}{2s}\Big(p_1 - p_2 + \tilde{x}(\lambda_1 u_1 - \lambda_2 u_2) + \tilde{y}(\lambda_1 v_1 - \lambda_2 v_2) + \tilde{z}(\lambda_1 w_1 - \lambda_2 w_2)\Big)$$

The terms s, and $\tilde{k}$ are defined as:

$$s = \sqrt{\lambda_0{}^2 + \beta}, \quad \textit{and} \quad \tilde{k} = \frac{\xi_k}{\sqrt{\xi_x{}^2 + \xi_y{}^2 + \xi_z{}^2}}, \quad k = x, y, z$$

The pressure is computed by one of the following equations:

$$p = p_1 - \lambda_1\Big(\tilde{x}(u - u_1) + \tilde{y}(v - v_1) + \tilde{z}(w - w_1)\Big) \tag{15}$$

or

$$p = p_2 - \lambda_2\Big(\tilde{x}(u - u_2) + \tilde{y}(v - v_2) + \tilde{z}(w - w_2)\Big) \tag{16}$$

In the above formulae λ_0, λ_1 and λ_2 are the eigenvalues. The viscous terms are discretized by central differences. The primitive variables at the cell face of the computational volume are calculated either from the left or from the right side according to the following formula:

$$U^{(l,r)}{}_{i+\frac{1}{2}} = \frac{1}{2}\Big((1 + sign(\lambda_l))U^{(l)}{}_{i+\frac{1}{2}} + (1 - sign(\lambda_l))U^{(r)}{}_{i+\frac{1}{2}}\Big) \tag{17}$$

where U is:

$$U = (p_l, u_l, v_l, w_l)^T$$

The characteristic variables at the cell faces are calculated by a third order interpolation scheme:

$$U^l_{i+\frac{1}{2}} = \frac{1}{6}(5U_i - U_{i-1} + 2U_{i+1}) \tag{18}$$

$$U^r_{i+\frac{1}{2}} = \frac{1}{6}(5U_{i+1} - U_{i+2} + 2U_i) \tag{19}$$

The time integration of the Navier-Stokes equations is obtained by an explicit Runge-Kutta scheme. The local time stepping technique is also used for steady state computations.

3. PARALLELIZATION ON BLOCK STRUCTURED GRIDS

3.1. Multiblock environment and data structure

The multiblock technique is used for flows in complex geometries where the grid generation on a single-block is very complicated. The geometry is subdivided into a number of blocks and the grid is generated on each of these blocks. The boundary conditions are stored into a layer of fictitious cells around the computational domain. Each block face can simultaneously contain different type of boundary conditions. The definition of several small blocks that contain homogeneous conditions on the faces is avoided. This is followed in order to reduce the load balancing effects on parallel computers, and therefore to achieve better performance.

3.2. Grid-partitioning and parallelization

A grid-partitioning algorithm for the decomposition of the computational domain into several subdomains was developed. Each block can be decomposed in one (1D-partitioning), two (2D-partitioning) or three directions (3D-partitioning), respectively. The partition strategy permits one or more geometrical blocks to be assigned per processor, and each geometrical block to be subdivided in several subdomains. Each processor can also contain more than one subdomains.

In the message passing model the computational grid is subdivided into non-overlapping subdomains, and each subdomain is assigned to one processor. Each processor stores its own data and these are exchanged between the processors during the numerical solution. In the shared memory model the data are stored in a common memory. Synchronization is obtained by synchronization primitive routines. In the present method local communication between the processors is needed after each Runge-Kutta iteration for updating the values along the subdomain boundaries. Global communication is needed for checking the convergence at each time step.

4. RESULTS

The present method was used for simulating laminar and turbulent flows over a sphere. Experiments for different Reynolds numbers have been performed in the past by Achenbach [6]. Three different grids were used with 760, 3,200, and 57,600 computational volumes, resprectively. A typical O-type grid over the sphere is shown in Figure 1. The flow was simulated for Reynolds numbers $Re = 1.62 \times 10^5$, and $Re = 5 \times 10^6$. In both

cases the flow is not axisymmetric. The flow structure in the wake of the sphere is shown in Figure 2 by plotting the streamlines. The pressure coefficient distributions are shown in Figure 3. For $Re = 1.62 \times 10^5$ the present results are compared with the corresponding predictions of reference [7] where an unstructured finite element scheme on a grid of 224,600 nodes had been used. In the present work the algebraic Baldwin-Lomax as well as the $k - \epsilon$ model were used for $Re = 5 \times 10^6$. The results for the pressure coefficient distribution are shown in Figure 3. It is clearly seen that $k - \epsilon$ model provides better results than the algebraic model. However, finer grids are still needed in order to improve the numerical predictions.

The above computations were performed on KSR1 and Convex SPP parallel systems. The parallelization on KSR1 was obtained by the shared memory (SM) as well as the message-passing model based on "local instruction" procedure (MPLI). The parallelization on Convex SPP was obtained by the PVM model. In Table 1 the total efficiency of the computations ($E_n = T_1/nT_n$, where T_1 and T_n are the computing times on one and n processors, respectively) is presented. An analysis of the total efficiency factor can be found in reference [8]. It is seen that SM and PVM models provide much better efficiency than the MPLI. The computing times are shown in the Table 2 for computations on eight processors. Convex SPP provides faster solution than KSR1 due to the faster processor. The effect of the grid refinement on the total efficiency is also shown in Table 2 for computations performed on Convex SPP and KSR1 systems, respectively. The performance of the parallel computations is significantly better on the finest grid.

5. CONCLUSIONS

A parallel three-dimensional upwind method was presented. The method was applied for simulating the flow over a sphere at different Reynolds numbers. The results are satisfactory for the laminar case, but in the higher Reynolds number case the grid used seems to be very coarse for resolving accurately the flow field. The $k - \epsilon$ model improves the prediction of the pressure coefficient distribution in comparison with the algebraic model. However, for the turbulent flow case the results are still far from the experimental measurements. Investigation of the unsteady behaviour of the flow over the sphere using different turbulence models is a future task. Parallelization of the method was obtained using shared memory and message passing models. The shared memory model provides better efficiency than the message-passing one on the KSR1 system. This is propably due to the hardware characteristics of the KSR1 system and it cannot be generalized as conclusion before an investigation on other parallel systems using also more processors is performed. The PVM model on the Convex SPP system was also used. The computing times on Convex SPP are much better than those on the KSR1. This is due to the faster processor of the Convex system.

REFERENCES

1. F. H. Harlow and J. E. Welch, Physics of Fluids, 8, (1965) 2182.
2. A. J. Chorin, J. of Comp. Phys., 2, (1967) 12.
3. V. C. Patel, W. Rodi and G. Scheuerer, AIAA J., 23, (1985) 1308.
4. B. S. Baldwin and H. Lomax, AIAA Paper 78-257, (1978).
5. D. Drikakis, P. Govatsos and D. Papantonis, Int. J. Num. Meth. Fluids, 19, (1994) 667.
6. E. Achenbach, J. Fluid Mech., 54, (1972) 565.
7. W. Koschel, M. Lötzerich, and A. Vornberger, AGARD Conf. Proc. No. 437, 1, (1988) 26-1.
8. D. Drikakis, E. Schreck, and F. Durst, J. Fluids Eng., 116, (1994) 835.

Table 1
Comparison of the total efficiencies using different parallelization models

	KSR1 - SM	KSR1 - MPLI	Convex SPP - PVM
1 proc.	100	100	100.
2 procs.	99.1	75.0	99.0
3 procs.	97.0	73.8	79.0
4 procs.	94.6	72.8	77.0
5 procs.	83.6	57.6	75.0
6 procs.	81.3	54.2	73.0
8 procs.	75.0	37.0	65.0

Table 2
Comparison of the computing time (sec./iter.) and total efficiencies on eight processors for three different grids (CV: computational volumes)

	KSR1 - SM	KSR1 - MPLI	Convex SPP - PVM
	Time, (E_n%)	Time, (E_n%)	Time, (E_n%)
760 CV	5.28 (23.0)	5.1 (23.7)	0.43 (27)
3,200 CV	10.30 (43.3)	15.0 (29.8)	0.66 (41)
57,600 CV	43.90 (75.0)	89.0 (37.0)	2.92 (65)

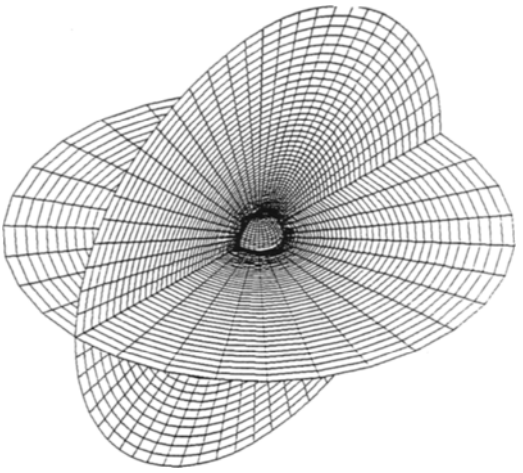


Figure 1: O-type grid around the sphere.

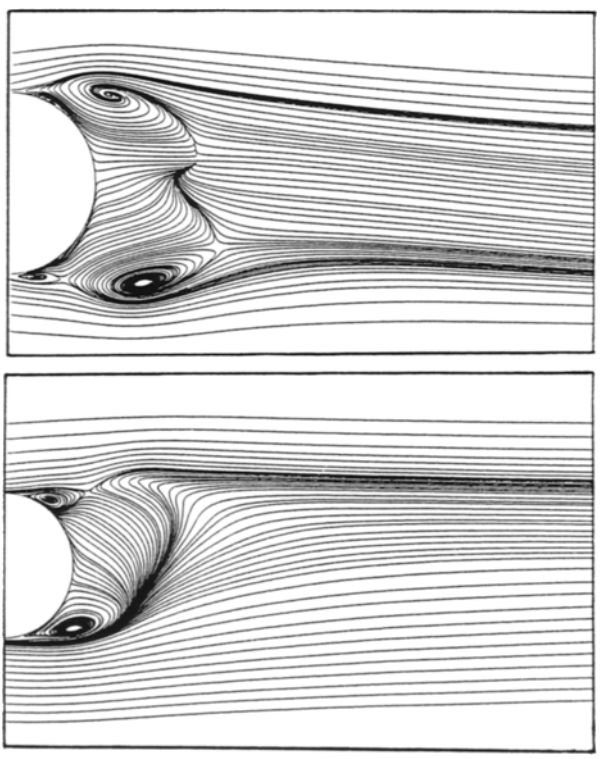


Figure 2: Flow structure in the wake of the sphere for $Re = 1.62 \times 10^5$ (top figure) and $Re = 5 \times 10^6$ (bottom figure).

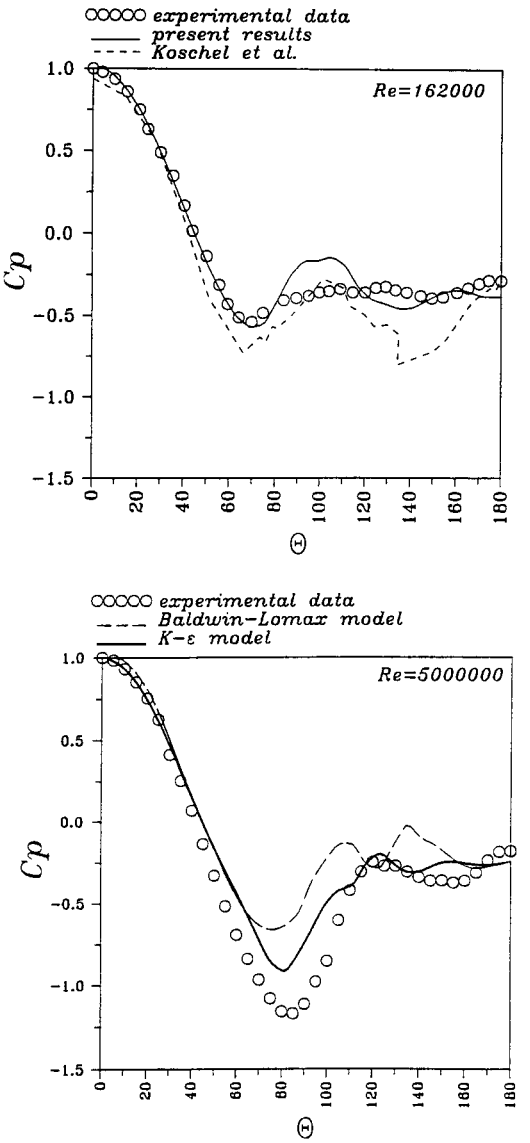


Figure 3: Pressure coefficient distributions for the flow over the sphere; for $Re = 1.62 \times 10^5$ (top figure) and $Re = 5 \times 10^6$ (bottom figure).

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

High order Padè-type approximation methods for incompressible 3D CFD problems on massively parallel computers

V.A. Garanzha[a], I.V. Ibragimov[a], I.N. Konshin[a], V.N. Konshin[a], and A.Yu. Yeremin[b]

[a]Computing Center of Russian Academy of Sciences,
Vavilova Str. 40, Moscow 117967, RUSSIA

[b]Institute of Numerical Mathematics of Russian Academy of Sciences,
Leninskij Prosp. 32a, Moscow 117334, RUSSIA

Abstract. We report results of numerical experiments with high order upwind Padè-type schemes for solving 3D incompressible CFD problems on massively parallel computers. It is shown that the high order schemes allow for the efficient massively parallel implementation with no damage to the high accuracy of the computed CFD solution. The results of numerical experiments with the 3D Dow Chemical stirred tank reactor problem are presented.

1. INTRODUCTION

Further progress in solution methods for challenging industrial CFD problems is mostly determined by the ability of efficient exploitation of massively parallel computers. The principal target when designing such methods is to get a high parallel efficiency with no damage to their mathematical properties. Any compromise between the overall parallel efficiency (measured in MFLOPS rates) and the mathematical properties (measured in terms of convergence, of arithmetic costs, of spectral resolution, of approximation quality and etc.) may be heavily problem dependent, hardware dependent and problem size dependent. Thus why the resulting mathematical method may lose the reliability which is one of the most important properties when dealing with industrial applications.

Generally speaking nowadays there exist two main approaches to constructions of mathematical methods for solving 3D CFD problems on massively parallel computers. The first approach is based on exploitation of low order approximation schemes on unstructured meshes. It allows for relatively simple implementations when dealing with complex geometrical configurations and requires essentially local mesh refinements in the regions of sharp gradients to obtain a reasonable accuracy. However, this approach requires solution of very large sparse arbitrarily populated (unstructured) but relatively well conditioned coefficient matrices frequently with a diagonal dominance. The second approach is based on exploitation of high order approximation schemes on almost

structured meshes. In contrast to the first approach this one provides the spectral resolution and the high accuracy in the whole computational domain (even using relatively coarse grids). The second approach requires solution of linear systems with large sparse structured but ill-conditioned coefficient matrices with a large off-diagonal dominance.

The first approach is mostly popular nowadays but potentially is less attractive since it requires a sophisticated tuning of the unstructured mesh, a considerable improvement in accuracy and also puts severe constraints onto the resulting parallel efficiency since it has to deal with large sparse unstructured matrices. The second approach is less popular due to the lack of reliable and efficient parallel iterative solution methods for ill-conditioned linear systems and due to severe problems when constructing the coefficient matrices. However, since it allows to deal with structured sparse matrices of smaller sizes, provides a high accuracy and the spectral resolution and possesses enough resources of potential parallelism we can expect an efficient implementation of high order approximation schemes on massively parallel computers for solving challenging industrial applications.

In this paper we report results of numerical experiments with high order Padè-type approximations on the massively parallel supercomputer CRAY T3D when solving the 3D incompressible stirred tank reactor problem.

2. HIGH ORDER SCHEME BASED ON UPWIND PADÉ-TYPE DIFFERENCES

Conventional finite difference approximations to differential operators may be represented as polynomial functions of discrete shift operators while the Padé-type or compact schemes are based on rational functions. Generally rational approximations give rise to more accurate and more stable but relatively simple difference schemes. Moreover, such schemes provide a better representation of the shorter length scales thus ensuring the so-called spectral-like resolution property.

In this paper we use new versions of the Padé-type differences [1] which retain all advantages of compact schemes and possess the following additional properties:

- discrete conservation;
- upwinding based on the flux splitting;
- high accuracy and reliability in the case of complex geometries based on the geometric conservation property;
- stable incompressible $div - grad$ formulation with $O(h^5)$ stabilization terms (for the case when the velocity and the pressure are defined at the cell centers);
- positive definiteness of resulting discrete operators.

For the system of conservation laws

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}_1}{\partial x_1} + \frac{\partial \mathbf{F}_2}{\partial x_2} + \frac{\partial \mathbf{F}_3}{\partial x_3} = 0$$

we construct the Padé-type approximation in a finite volume fashion to the integral equality

$$\frac{\partial}{\partial t}\int\limits_{\Omega_k} \mathbf{Q}d\Omega + \int\limits_{\partial\Omega_k} (\mathbf{F}_1 n_1 + \mathbf{F}_2 n_2 + \mathbf{F}_3 n_3)ds = 0, \tag{1}$$

where Ω is a curvilinear mesh cell of a structured grid and $\mathbf{n}$ is the unit normal to $\partial\Omega_k$.

Values of fluxes $\mathbf{F}_i$ on appropriate cell faces are computed using upwind Padé-type interpolation operators consistent with centered Padé-type operators which are employed in computation of volume and surface integrals in (1).

In the Cartesian coordinates the formal approximation order of the upwind Padé-type finite differences is equal to $O(h^5)$ for inviscid terms of the conservation laws while in the case of curvilinear coordinates it decreases to $O(h^4)$ ensuring the geometric conservation property, i.e., the uniform flow is the exact solution to the resulting system of discrete equations.

The cost of the Padé-type differences in right-hand sides computation, regardless of the number of space dimensions, involves only inversions of the narrow banded (usually tridiagonal) matrices, and hence is comparable to conventional finite differences.

For solving the resulting systems of nonlinear discrete equations we exploit fully implicit predictor-corrector type time integration techniques or Newton-type iterative methods. Both approaches require solution of large systems of linear algebraic equations but with positive definite coefficient matrices. The nonlocality of the Padé-type approximation schemes formally leads to dense coefficient matrices (Jacobians) but allows for very sparse multiplicative presentations of these dense coefficient matrices. We develop a two stage procedure for constructing high quality preconditioners. At the first stage dense coefficient matrices are approximated in the spectral sense by sparse matrices which are approximated at the second stage by well suited to massively parallel computers incomplete block triangular preconditioners described in Section 3.

Numerical experiments with model problems and mesh refinement studies [1] reveals that presented schemes and boundary conditions treatment provide drastic improvement in accuracy over conventional finite differences. Numerical evidences indicate that this technique provides the realistic $O(h^4)$ accuracy on curvilinear nonorthogonal meshes and very accurate solutions can be obtained on coarse meshes.

3. MASSIVELY PARALLEL ITERATIVE SOLUTION OF LARGE SPARSE LINEAR SYSTEMS

When choosing the iterative solution strategy for large sparse ill-conditioned linear systems originated from high order Padè-type schemes we tried to avoid first of all any compromise between the parallel properties of the iterative method and its serial arithmetic complexity. To this end we consider incomplete block factorization preconditioning strategies with factorized sparse approximate inverses to approximate pivot block [2].

Let A be a $n \times n$ block matrix of the form

$$A = L + D + U,$$

where D is the block diagonal part of A, $D = Block\,Diag\,(D_1, \ldots, D_n)$, L and U are, respectively, the block lower and upper strictly triangular parts of A. The incomplete block factorization of A in the inverse free form is then constructed as follows.

$$A \approx (G^{-1} + L)G(G^{-1} + U) = (I + LG)G^{-1}(I + GU),$$

where G is a block diagonal matrix whose blocks are determined using the recurrence relations

$$G_i = \Omega_i(X_i^{-1}), \quad i \geq 1, \quad X_1 = D_1,$$

$$X_i = D_i - A_{ii-1}G_{i-1}A_{i-1i}, \quad i \geq 2,$$

and $\Omega_i(X_i^{-1})$ denotes a sparse approximation to the inverse of X_i.

To construct $\Omega_i(X_i^{-1})$ we use factorized sparse approximate inverses technique. In this case we construct sparse approximate inverses $G_i^{(L)}$ and $G_i^{(U)}$ to the exact triangular factors of $X_i = L_iU_i$ by minimizing the matrix Frobenious norms

$$\| I - L_iG_i^{(L)} \|_F \text{ and } \| I - G_i^{(U)}U_i \|_F .$$

Here $G_i^{(L)}$ and $G_i^{(U)}$ are parametrized only by locations of nonzero entries, their construction does not require any information about entries of L_i and U_i and consists of solution of many independent small size linear systems (one linear system to compute each row of $G_i^{(L)}$ and $G_i^{(U)}$) whose size are equal to the numbers of nonzero entries in the corresponding rows of $G_i^{(L)}$ and $G_i^{(U)}$. Hence, computation of an incomplete block factorization preconditioner possesses by construction a large resources of natural parallelism. Moreover, multiplication of an incomplete block factorization preconditioner by a vector also possesses by construction a large resources of natural parallelism since it requires only solutions of block triangular matrices with identity blocks on the main diagonal and multiplications of G by a vector.

The results of numerical experiments [3] on the massively parallel computer CRAY T3D with incomplete block factorization preconditioned GMRES method have demonstrated the quite reasonable scalability when solving very large linear systems originated from 3D CFD problems.

4. NUMERICAL EXPERIMENTS

In this section we present results of numerical experiments with high order Padè-type approximations on the massively parallel supercomputer CRAY T3D. For the comparison purposes we also present the corresponding results on 1 CPU of the vector/parallel supercomputer CRAY C90. As a sample real world problem we have chosen the 3D laminar Dow Chemical stirred tank reactor problem [4]. We considered a silicon flow in a cylindrical tank with four vertical baffles and the pitched blade turbine (PBT) agitator. In this regime ($100rpm$, silicon) the Reynolds number computed by the impeller radius, the velocity at the impeller tip and the silicon viscosity is about

150.

The sample problem was approximated on the $81 \times 81 \times 40$ curvilinear mesh. Thus at each Newton iteration we solved a linear system of size $N = 1.049.760$. Only $6 - 7$ Newton iterations were sufficient to reduce the nonlinear residual up to the machine precision.

Tables 1 and 2 contain the timing and the performance results when solving the sample problem on T3D and C90. The only one difference in between data of Tables 1 and 2 is related to multiplications of the Jacobians by vectors. We remind that when multiplying the Jacobian by a vector we need to solve several auxiliary block tridiagonal linear systems to avoid the explicit generation of the exact Jacobian (and thus in a sense "to regenerate" the problem). So in Table 1 we include these costs into the problem generation phase while in Table 2 we include them into iterative solution phase. Tables 1 and 2 adopt the following notation: **NCPUS** denotes the number of multiple CPU's, **TIME** stands for the total CPU time in seconds for solving the sample stirred tank reactor problem, **Perf** denotes the sustained performance in MFLOPS, the subscripts "**g**" and "**s**" stand for the linear system generation phase and the iterative solution phase, respectively, and **Speedup** denotes the actual speedup obtained using multiple CPU's of T3D as compared with 1 CPU of C90.

Table 1
Timing and Performance Characteristics of Padè-type Approximations on CRAY T3D

	T3D				C90
NCPUS	16	32	64	128	1
TIME	3220	1623	867	444	1320
$TIME_g$	1912	958	512	266	792
$TIME_s$	1308	665	355	178	528
$Perf_g$	111	211	410	795	267
$Perf_s$	156	304	592	1152	396
Perf	129	249	485	938	318
Speedup	0.41	0.78	1.53	2.95	1.00

Table 2
Timing and Performance Characteristics of Padè-type Approximations on CRAY T3D

	T3D				C90
NCPUS	16	32	64	128	1
TIME	3220	1623	867	444	1320
$TIME_g$	60	33	18	10	27
$TIME_s$	3160	1590	849	434	1293
$Perf_g$	111	211	410	795	267
$Perf_s$	130	250	487	943	318
Perf	129	249	485	938	318
Speedup	0.41	0.78	1.53	2.95	1.00

Figure 1 shows the comparison of the computed and the experimental velocity fields for the sample stirred tank reactor problem.

We did not try to simulate the flow around impeller, instead of that the time-averaged experimental data at the surface surrounding the impeller were used as boundary conditions. Nevertheless, reasonable agreement with experimental data is achieved.

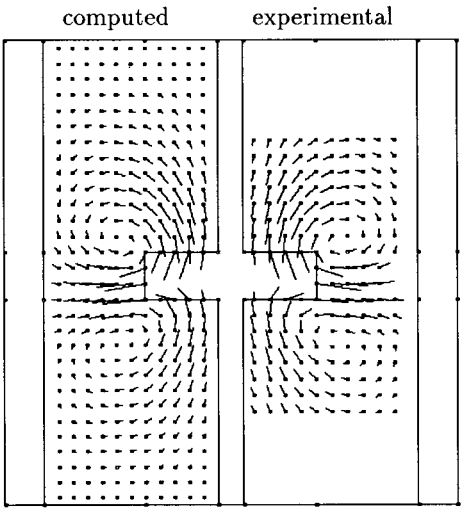


Figure 1. Velocity fields for the laminar stirred tank problem.

Tables 1 and 2 show that high order Padè-type approximations allow efficient implementations on vector and massively parallel computers. Here we emphasize that we ran precisely the same mathematical algorithms on 1 CPU of C90 and on multiple CPU's of T3D.

Acknowledgments. The authors are grateful to CRAY Research, Inc. (USA) for providing the computer resources.

REFERENCES

1. V.A. Garangha and V.N. Konshin, Highly Accurate Upwind Padè-type Approximations for the Systems of Conservation Laws. I: Application to the Incompressible Navier-Stokes Equations on a Single-Zone Structured Meshes, Elegant Mathematics, Inc. (USA), Research Report EM-RR-21/95.
2. L.Yu. Kolotilina and A.Yu.Yeremin, Incomplete Block Factorizations as Preconditioners for Sparse SPD Matrices, Research Report EM-RR-6/92, Elegant Mathematics, Inc.(USA), 1992.

3. I.V. Ibragimov, P.A. Kolesnikov, I.N. Konshin, N.E. Mikhailovsky, A.A. Nikishin, E.E. Tyrtyshnikov, and A.Yu. Yeremin, Numerical Experiments with Industrial Iterative Solvers on Massively Parallel Computers. I: Numerical Experiments with the A_SPARSE Solver on CRAY T3D, High Performance Computer Conference, Phinics. Arizona, USA, pp.283-289 (1995).
4. S. Sen, C.K. Lee, and D.E. Leng, Modelling of Turbulence in Stirred Tanks with Experimental Validation, Paper 189a, 1994 Annual AIChE Meeting, San Francisko, Nov. 1994.

This Page Intentionally Left Blank

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel Computations of CFD Problems Using a New Fast Poisson Solver

Masanori Obata and Nobuyuki Satofuka

Department of Mechanical and System Engineering, Kyoto Institute of Technology
Matsugasaki, Sakyo-ku, Kyoto 606, JAPAN

A fast Poisson solver based on the rotated finite difference discretization is applied for solving two-dimensional and three dimensional problems. The method is implemented for two-dimensional problems on a cluster of workstations using PVM, and its convergence rate is compared with that of the conventional SOR method. The result shows 6.5 times faster convergence speed for two-dimensional test case with the same accuracy.

1. INTRODUCTION

Many fluid dynamic problems of fundamental importance in the field of computational fluid dynamics are governed by the elliptic partial differential equations (Poisson equation). Especially, in the case of solving time dependent incompressible viscous flow problems, the Poisson equation for either the stream function or the pressure have to be solved at each time step. In such case, most part of computational time is spent in solving the Poisson equation. Iterative methods are generally used as suitable approach to the solution of the Poisson equation. The successive overrelaxation (SOR) method[1] is one of the most popular iterative method to solve the Poisson equation and a number of modifications, e.g. line SOR, group SOR has been proposed to accelerate the convergence. The group explicit iterative (GEI) method[2] is proposed not only to improve the convergence rate but also to reduce the computational cost. The method is easily adapted to architecture of recent advanced computers using vector and/or parallel processing. The multigrid method[3] is used to accelerated further the convergence of those iterative method.

In this paper, we propose a new efficient iterative method for solving the Poisson equation based on the rotated finite difference approximation. In 2-D and 3-D problems, the convergence history and the accuracy are compared with the SOR method with conventional finite difference approximation. In 2-D case, the multigrid method is also applied for solving the Poisson equation. The parallelization is carried out on a cluster of networked workstations. Finally, those Poisson solvers are applied to solve 2-D driven cavity flow at Re=100.

2. BASIC CONCEPT

The Poisson equation for ϕ with the source term $f(x, y, z)$ in the three-dimensional Cartesian coordinate is written as,

$$\nabla^2\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = f(x,y,z). \tag{1}$$

We consider Eq.(1) as a model equation for applying the present methods.

2.1 Rotated finite difference approximation method

Equation (1) is discretized using the nine-point rotated finite difference approximation, which is given by

$$\begin{aligned}\nabla^2\phi_{i,j,k} = \frac{1}{4h^2}(&\phi_{i-1,j-1,k-1} + \phi_{i+1,j-1,k-1} + \phi_{i-1,j+1,k-1} + \phi_{i+1,j+1,k-1} \\ &+ \phi_{i-1,j-1,k+1} + \phi_{i+1,j-1,k+1} + \phi_{i-1,j+1,k+1} + \phi_{i+1,j+1,k+1} - 8\phi_{i,j,k}) \equiv f_{i,j,k}\end{aligned} \tag{2}$$

where h denotes grid spacing, defined as,

$h = \Delta x = \Delta y = \Delta z$.

and i, j and k denote grid indices.

2.2 Successive overrelaxation method with rotated finite difference approximation

Equation (1) with the successive overrelaxation (SOR) method (RFD-SOR) is written by using the rotated finite difference approximation, as follows,

$$\begin{aligned}\phi^{n+1}_{i,j,k} = (1-\omega_{SOR})\phi^{n}_{i,j,k} + \frac{\omega_{SOR}}{8}(&\phi^{n+1}_{i-1,j-1,k-1} + \phi^{n+1}_{i+1,j-1,k-1} + \phi^{n+1}_{i-1,j+1,k-1} + \phi^{n+1}_{i+1,j+1,k-1} \\ &+ \phi^{n}_{i-1,j-1,k+1} + \phi^{n}_{i+1,j-1,k+1} + \phi^{n}_{i-1,j+1,k+1} + \phi^{n}_{i+1,j+1,k+1} - 4h^2 f_{i,j,k})\end{aligned} \tag{3}$$

where n denotes a number of iteration step. Since only diagonal points are used in this discretization, the grid points are classified into four colors in 3-D. The computation on each group of colored grid points can be carried out independently. Only a quarter of the grid points in the computational domain are iterated until convergence. The solution of the remaining points are obtained with seven-point two-dimensionally rotated finite difference approximation. For example, $\phi_{i,j,k+1}$ is obtained from x-y rotated approximation as,

$$\phi_{i,j,k+1} = \frac{1}{8}(\phi_{i-1,j-1,k+1} + \phi_{i+1,j-1,k+1} + \phi_{i-1,j+1,k+1} + \phi_{i+1,j+1,k+1} + 2\phi_{i,j,k} + 2\phi_{i,j,k+2} - 2h^2 f_{i,j,k}). \tag{4}$$

In two-dimensional case, the grid points are classified into two colors, as the same as red-black ordering. After the solution of a group has converged, the remaining points are obtained from conventional five-point finite difference approximation. The correction cycle algorithm is used for the multigrid strategy.

2.3 Successive overrelaxation method with explicit decoupled group method

The explicit decoupled group (EDG) algorithm[4] is also introduced. We can consider eight-points group, such as,

$$\begin{bmatrix} 8 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 8 \end{bmatrix} \begin{bmatrix} \phi_{i,j,k} \\ \phi_{i+1,j+1,k+1} \\ \phi_{i+1,j,k} \\ \phi_{i,j+1,k+1} \\ \phi_{i+1,j+1,k} \\ \phi_{i,j,k+1} \\ \phi_{i,j+1,k} \\ \phi_{i+1,j,k+1} \end{bmatrix}$$

$$= \begin{bmatrix} -4h^2 f_{i,j,k} + \phi_{i-1,j-1,k-1} + \phi_{i+1,j-1,k-1} + \phi_{i-1,j+1,k-1} + \phi_{i+1,j+1,k-1} + \phi_{i-1,j-1,k+1} + \phi_{i+1,j-1,k+1} + \phi_{i-1,j+1,k+1} \\ -4h^2 f_{i+1,j+1,k+1} + \phi_{i+2,j,k} + \phi_{i,j+2,k} + \phi_{i+2,j+2,k} + \phi_{i,j,k+2} + \phi_{i+2,j,k+2} + \phi_{i,j+2,k+2} + \phi_{i+2,j+2,k+2} \\ -4h^2 f_{i+1,j,k} + \phi_{i,j-1,k-1} + \phi_{i+2,j-1,k-1} + \phi_{i,j+1,k-1} + \phi_{i+2,j+1,k-1} + \phi_{i,j-1,k+1} + \phi_{i+2,j-1,k+1} + \phi_{i,j+1,k+1} \\ -4h^2 f_{i,j+1,k+1} + \phi_{i-1,j,k} + \phi_{i-1,j+2,k} + \phi_{i+1,j+2,k} + \phi_{i-1,j,k+2} + \phi_{i+1,j,k+2} + \phi_{i-1,j+2,k+2} + \phi_{i+1,j+2,k+2} \\ -4h^2 f_{i+1,j+1,k} + \phi_{i,j,k-1} + \phi_{i+2,j,k-1} + \phi_{i,j+2,k-1} + \phi_{i+2,j+2,k-1} + \phi_{i+2,j,k+1} + \phi_{i,j+2,k+1} + \phi_{i+2,j+2,k+1} \\ -4h^2 f_{i,j,k+1} + \phi_{i-1,j-1,k} + \phi_{i+1,j-1,k} + \phi_{i-1,j+1,k} + \phi_{i+1,j-1,k+2} + \phi_{i+1,j-1,k+2} + \phi_{i-1,j+1,k+2} + \phi_{i+1,j+1,k+2} \\ -4h^2 f_{i,j+1,k} + \phi_{i-1,j,k-1} + \phi_{i+1,j,k-1} + \phi_{i-1,j+2,k-1} + \phi_{i+1,j+2,k-1} + \phi_{i-1,j,k+1} + \phi_{i-1,j+2,k+1} + \phi_{i+1,j+2,k+1} \\ -4h^2 f_{i+1,j,k+1} + \phi_{i,j-1,k} + \phi_{i+2,j-1,k} + \phi_{i,j+1,k} + \phi_{i+2,j-1,k+2} + \phi_{i,j+1,k+2} + \phi_{i+2,j+1,k+2} \end{bmatrix} \tag{5}$$

From Eq.(5), we can obtain 4 decoupled system, and one of them is

$$\begin{bmatrix} 8 & -1 \\ -1 & 8 \end{bmatrix} \begin{bmatrix} \phi_{i,j,k} \\ \phi_{i+1,j+1,k+1} \end{bmatrix}$$

$$= \begin{bmatrix} -4h^2 f_{i,j,k} + \phi_{i-1,j-1,k-1} + \phi_{i+1,j-1,k-1} + \phi_{i-1,j+1,k-1} + \phi_{i+1,j+1,k-1} + \phi_{i-1,j-1,k+1} + \phi_{i+1,j-1,k+1} + \phi_{i-1,j+1,k+1} \\ -4h^2 f_{i+1,j+1,k+1} + \phi_{i+2,j,k} + \phi_{i,j+2,k} + \phi_{i+2,j+2,k} + \phi_{i,j,k+2} + \phi_{i+2,j,k+2} + \phi_{i,j+2,k+2} + \phi_{i+2,j+2,k+2} \end{bmatrix}. \tag{6}$$

By introducing the SOR method (EDG-SOR), the iterative procedure for those points are obtained from following equations,

$$\begin{bmatrix} \phi^{*}_{i,j,k} \\ \phi^{*}_{i+1,j+1,k+1} \end{bmatrix}$$

$$= \frac{1}{63} \begin{bmatrix} 8 & 1 \\ 1 & 8 \end{bmatrix} \begin{bmatrix} -4h^2 f_{i,j,k} + \phi^{n+1}_{i-1,j-1,k-1} + \phi^{n+1}_{i+1,j-1,k-1} + \phi^{n+1}_{i-1,j+1,k-1} + \phi^{n+1}_{i+1,j+1,k-1} + \phi^{n+1}_{i-1,j-1,k+1} + \phi^{n+1}_{i+1,j-1,k+1} + \phi^{n}_{i-1,j+1,k+1} \\ -4h^2 f_{i+1,j+1,k+1} + \phi^{n}_{i+2,j,k} + \phi^{n}_{i,j+2,k} + \phi^{n}_{i+2,j+2,k} + \phi^{n}_{i,j,k+2} + \phi^{n}_{i+2,j,k+2} + \phi^{n}_{i,j+2,k+2} + \phi^{n}_{i+2,j+2,k+2} \end{bmatrix},$$

$$\begin{bmatrix} \phi^{n+1}_{i,j,k} \\ \phi^{n+1}_{i+1,j+1,k+1} \end{bmatrix} = (1-\omega_{SOR}) \begin{bmatrix} \phi^{n}_{i,j,k} \\ \phi^{n}_{i+1,j+1,k+1} \end{bmatrix} + \omega_{SOR} \begin{bmatrix} \phi^{*}_{i,j,k} \\ \phi^{*}_{i+1,j+1,k+1} \end{bmatrix} \tag{7}$$

In 2-D case, we can consider four-point group, and one of the decoupled system is solved as,

$$\begin{bmatrix} \phi^{*}_{i,j} \\ \phi^{*}_{i+1,j+1} \end{bmatrix} = \frac{1}{15}\begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix}\begin{bmatrix} -2h^2 f_{i,j} + \phi^{n+1}_{i-1,j-1} + \phi^{n+1}_{i+1,j-1} + \phi^{n+1}_{i-1,j+1} \\ -2h^2 f_{i+1,j+1} + \phi^{n}_{i+2,j} + \phi^{n}_{i,j+2} + \phi^{n}_{i+2,j+2} \end{bmatrix}$$

$$\begin{bmatrix} \phi^{n+1}_{i,j} \\ \phi^{n+1}_{i+1,j+1} \end{bmatrix} = \omega_{SOR}\begin{bmatrix} \phi^{*}_{i,j} \\ \phi^{*}_{i+1,j+1} \end{bmatrix} + (1-\omega_{SOR})\begin{bmatrix} \phi^{n}_{i,j} \\ \phi^{n}_{i+1,j+1} \end{bmatrix} \quad . \tag{8}$$

After the solution of a colored group has converged, the remaining points are obtained from conventional five-point finite difference approximation. The correction cycle algorithm is used for the multigrid strategy. As the standard EDG method composed even-point groups, the method is difficult to combine with the multigrid acceleration. To avoid this difficulty, the computational domain is divided into four blocks, as shown in Fig. 1. Only the filled points is iterated by using the connected group. The gird points in hatched area is swept at each block computation.

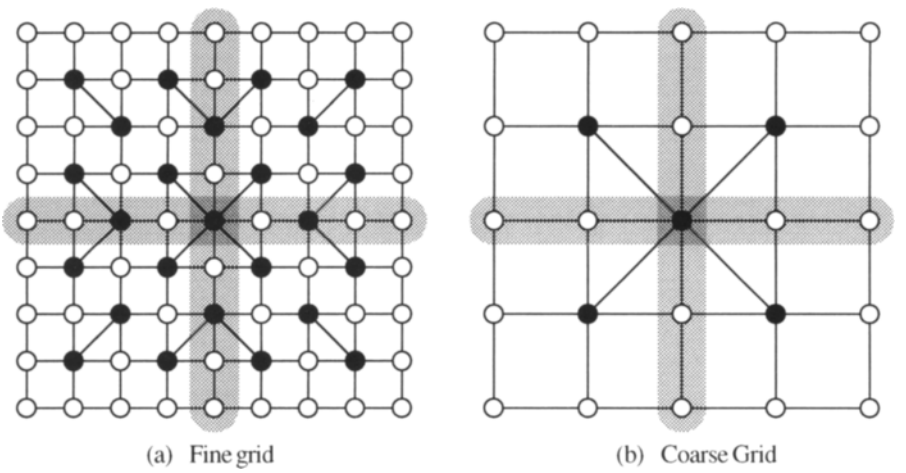


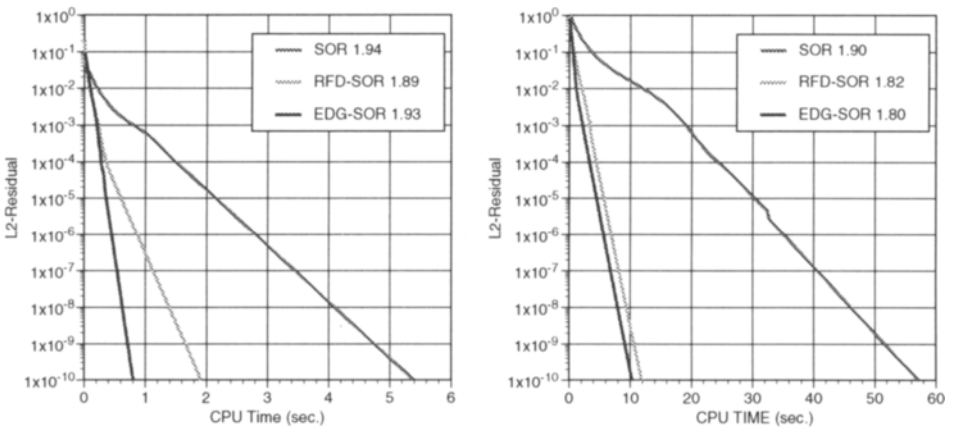
Figure 1. EDG method with multigrid algorithm

3. PARALLELIZATION

A domain decomposition technique for the macro-level red-black SOR method[5] is applied for solving incompressible viscous flows on networked engineering workstations. The message passing is handled by PVM (Parallel Virtual Machine) software. For the parallel computations using PVM, networked workstations of HP9000 model 720 are used. In the parallelization strategy the domain decomposition approach is used in which the whole physical domain is divided into a number of smaller subdomains. One line of overlapping auxiliary grids surrounding a given subdomain is used to make available the information required in the determination of flow variables at the block boundary. Basically each subdomain is treated by a different processor. During the numerical integration of the equations the residuals of each subdomain is checked at each iteration step. When the residuals of entire domain drop below a prescribed limit, the computation is stopped.

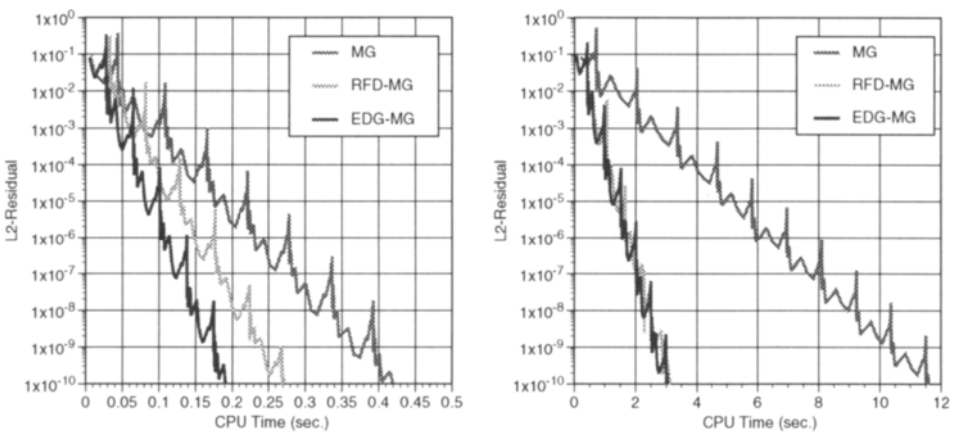
4. TEST PROBLEMS

At first, test cases with $f(x,y)$=-sin$x$-sin$y$ in 2-D and $f(x,y,z)$=-sin$x$-sin$y$-sin$z$ in 3-D are solved. The convergence criterion is, $L2-Residual<1.0\times10^{-10}$. The convergence history of L2-Residual at the optimum overrelaxation factor for 2-D and 3-D test problems are shown in Fig.2 (a) and (b). "RFD-SOR" means the rotated finite difference method with SOR, and "EDG-SOR", EDG with SOR. The optimum overrelaxation factor ω_{SOR} is also shown. The abscissa is the CPU time on Kubota TITAN II-400. In both cases, the EDG algorithm is the fastest one among them for solving the Poisson equation. In the next, the multigrid method is combined with those method to solve the Poisson equation. Figure 3 (a) and (b) shows the history of L2-Residual. Again the EDG algorithm is the fastest.



(a) 2-D (128x128 Grid) (b) 3-D (64x64x64 Grid)

Figure 2 History of L2-Residual for SOR Method



(a) 2-D (129x129 Grid) (b) 3-D (65x65x65 Grid)

Figure 3. History of L2-Residual for multigrid method

Figure 4 (a) and (b) show the L2-Error of the those methods for 2-D and 3-D test cases, respectively. The figures indicate that those methods have the second order accuracy in 2-D and 3-D. Finally, Figure 5 shows the speedup using PVM on the cluster of workstations. The macro-level red-black SOR[5] is used for the parallelization of the SOR method, the RFD-SOR method and the EDG-SOR.

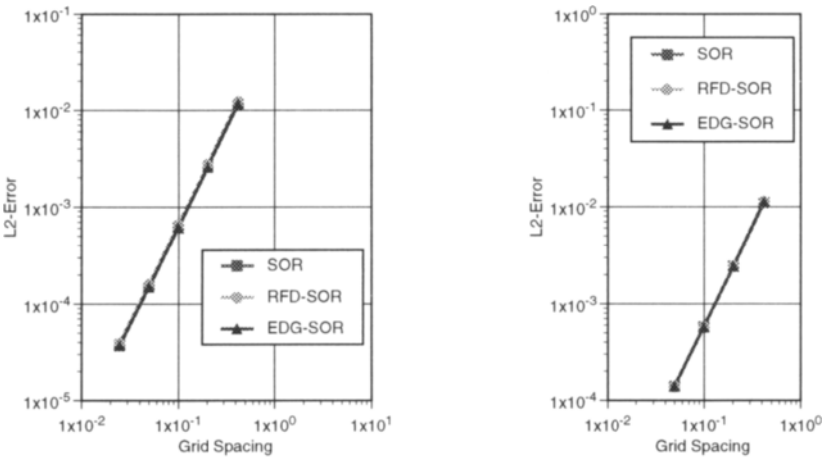


Figure 4. Spatial Accuracy

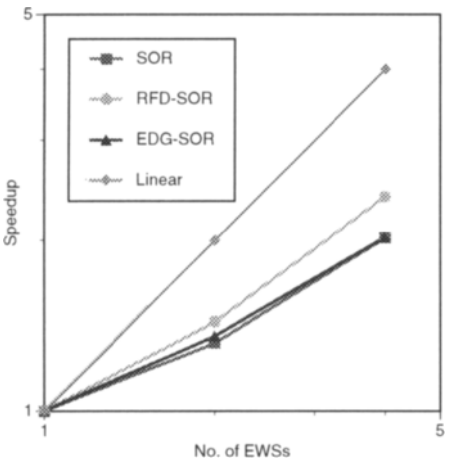


Figure 5. Speedup of two-dimansional computation on 258x258 Grid

5. APPLICATION TO TWO-DIMENSIONAL DRIVEN CAVITY FLOW

5.1 Incompressible Navier-Stokes equations and numerical procedure

With the vorticity ω and stream function ψ, the Navier-Stokes equations are written as,

$$\frac{\partial \omega}{\partial t}+\frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x}-\frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y}=\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 \omega}{\partial x^2}+\frac{\partial^2 \omega}{\partial y^2}\right), \tag{9}$$

$$\frac{\partial^2 \psi}{\partial x^2}+\frac{\partial^2 \psi}{\partial y^2}=-\omega, \tag{10}$$

where the vorticity ω and stream function ψ are defined as

$$\omega=\frac{\partial v}{\partial x}-\frac{\partial u}{\partial y}, \tag{11}$$

$$\frac{\partial \psi}{\partial x}=-v, \quad \frac{\partial \psi}{\partial y}=u, \tag{12}$$

Equations (9) and (10) are solved using a method of lines approach with the conventional second-order accurate central finite difference approximation and the second-order accurate two-stage rational Runge-Kutta (RRK) time integration scheme[6].

5.2 Numerical result

A driven cavity flow with Reynolds number Re=100 is solved. The results obtained by using the SOR method, the rotated finite difference method and the EDG method for solving Eq. (10) are compared. Figure 6 (a) shows the vorticity contours, and Fig 6 (b), the streamline with 130x130 grid points. The solution of each method are in good agreement with each other. Figure 7 shows the convergence history with 130x130 grid points. The abscissa is the CPU time on Kubota TITAN II-400. The EDG-SOR method is the fastest among those methods. Figure 8 shows the speedup on networked workstations with 514x514 grid points. The speedup of rotated finite difference and EDG are 2.4 and 2.0 because of high latency of the Ethernet.

6. CONCLUSIONS

New fast Poisson solver are proposed and applied to solve incompressible Navier-Stokes equations. The following conclusions are drawn,

(1) In 2-D and 3-D test problems for Poisson equations, the CPU time of the RFD- and EDG-SOR method is reduced to 37% and 15% of the SOR method in 2-D case, and 21% and 18% in 3-D case. The accuracy of the RFD- and EDG-SOR method is comparable to the SOR method.

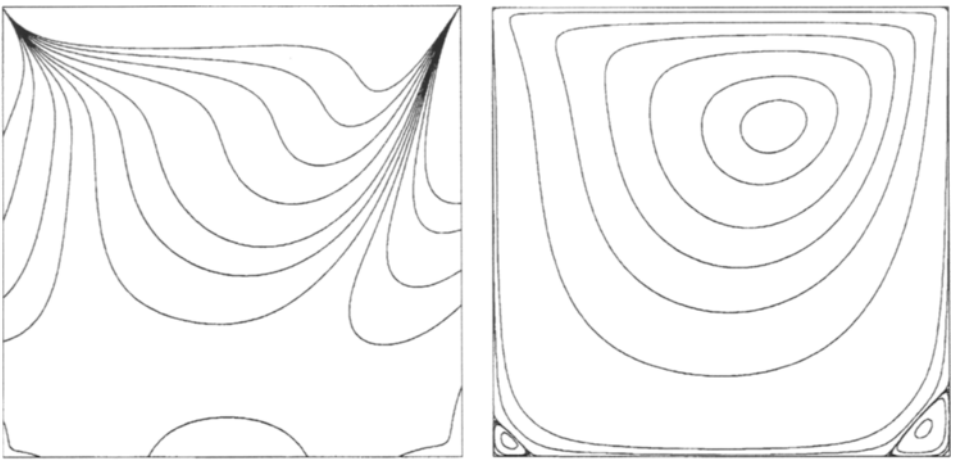
(2) In 2-D test problems for Poisson equation, the speedup of the RFD- and EDG-SOR method parallelized on network of 4 workstations using PVM is 2.4 and 2.0, respectively, because of high latency.

(3) In computations of the 2-D driven cavity flow, the CPU time of RFD- and EDG-SOR method is reduced to 35% and 26% of the SOR method. The speedup of the RFD- and EDG-SOR method on network of 4 workstations is 1.76 and 1.69.

REFERENCES

1. D. Young, Iterative Methods for Solving Partial Differential Equations of Elliptic Type, Trans. Amer. Math. Soc., Vol. 76, pp. 92-111 (1954)

2. D.J. Evans, Group Explicit Iterative Methods for Solving Large Linear System, Int'l. J. Comput. Math., Vol. 17, pp. 81-108 (1985)



(a) Voriticity (b) Streamline function

Figure 6. Contours of two-dimensional driven cavity flow
solid;SOR, dotted;RFD-SOR, dashed;EDG-SOR

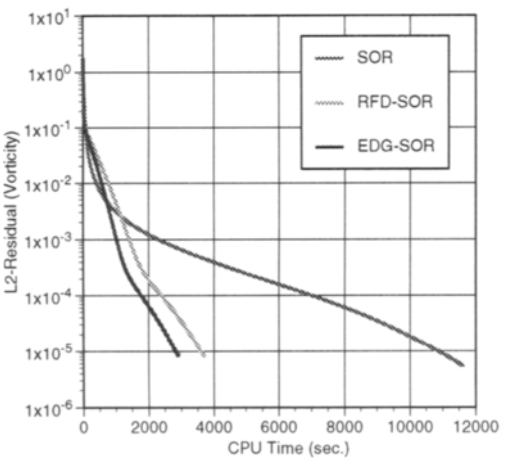


Figure 7. History of L2-Residual

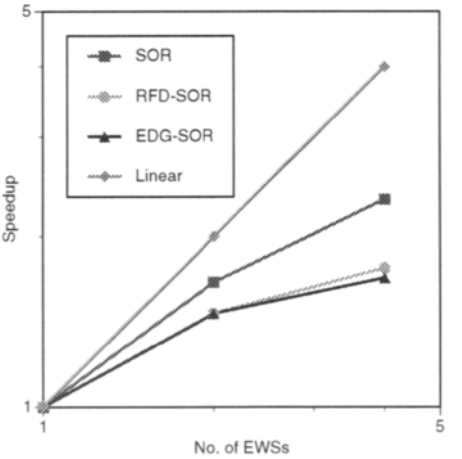


Figure 8. Speedup on 514x514 Grid

3. A. Brandt, Multi-Level Adaptive Solutions to Boundary-Value Problems, Math. Comput., Vol. 31, No. 138, pp. 333-390 (1977)
4. A. R. Abdullah, The Four Point Explicit Decoupled Group (EDG) Method: A Fast Poisson Solver, Int'l. J. Comput. Math., Vol. 38, pp. 61-70 (1991)
5. M. Obata, N. Satofuka and T. Suzuki, Domain Decomposition Technique for 2-D Incompressible Navier-Stokes solver on Array of Transputers, Proc. of Parallel CFD'94, to be published.
6. A. Wambecq, Rational Runge-Kutta Methods for Solving Systems of Ordinary Differential Equation, Computing, Vol. 20, pp. 333-342 (1978)

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

A Parallel Free Surface Flow Solver

J. A. Cuminato [a*]

[a]Department of Computer Sciences
University of São Paulo
P.O. Box 668, 13560-970 São Carlos - SP

This work describes a parallel technique for solving the Navier-Stokes equations in two dimensions with free surfaces. The parallel code is based on the GENSMAC one (Tome & McKee [1993-1994]) which implements the SMAC (Amsden & Harlow [1971]) method for the case of a general domain and general boundary conditions. The parallel version presented here comprises of three main parts: The momentum equation solver, The Possion solver and Particle movement. The momentum equation solver is based on explicit finite difference discretization of the original equations in primitive variables, the Possion solver is based on 5-point FD discretization of Possion's equation and the particles generated at the inlets move according to the velocity field calculated in the two previous steps. The parallelization is performed by splitting the computation domain into a number of vertical strips and assigning each of these to one processor. All the computation can then be performed using next neighbour communication only.

1. Introduction

The marker-and-cell (MAC) method, introduced by Harlow and Welsh [1965] is particularly suited for the modeling of fluid flows with free surfaces. One of its important features is the use of virtual particles which move from one cell to the next as a function of the flow velocities. If a cell contains a number of particles it is considered to contain fluid providing the localization of the free surface. It is thus, a special feature of the MAC method that the domain is split into a number of cells which are denoted to be either empty, full, surface or boundary cells, and one has to keep track of these cells as time evolves.

Amsden and Harlow [1971] developed a simplified MAC method (SMAC) where the calculation cycle is split into two parts: a provisional velocity field calculation followed by a velocity update using a potential function which is the solution of Poisson's equation and ensures mass conservation.

Tome and McKee [1994] developed this technique further by adding a number of specially designed devices to deal with general domains and multiple inlets/outlets. They called it GENSMAC. This code has been successfully used to simulate different types of 2D free surface flows, such as: jet filling of complex shaped cavities, penstock head-gate

*This work was partially undertaken during a visit to Strathclyde University sponsored by FAPESP under grant #93/3622-7

closure, jet buckling and die swell problems. However its generalization to cope with 3D problems is not feasible on a serial computer due to the staggering amount of particles needed to represent the 3D flow properly, let alone the difficulties in the visualization of the fluid boundaries. Another difficulty GENSMAC faces is in coping with slow flow of very viscous fluids because in this case stability restrictions make the time step very small, leading to unrealistic run times.

In this work we present modifications to the original SMAC technique implemented in GENSMAC and describe a parallel algorithm based on domain decomposition for the implementation of the SMAC method which does not present the same difficulties. For clarity, the presentation will be made for the 2D case only, but one should have no difficulty in generalizing it to cope with 3D problems.

2. The SMAC Method

The SMAC method is based on a staggered grid finite differences discretization of the equations arising from the following procedure used to solve the Navier-Stokes equations:

Assume that at time t_0 the velocity field $\mathbf{u}(\mathbf{x}, t_0)$ is given. Calculate $\mathbf{u}(\mathbf{x}, t)$, $t = t_0 + \Delta t$ from the following:

1. Let $\tilde{p}(\mathbf{x}, t_0)$ be an arbitrary pressure field satisfying the correct pressure conditions on the free surface.

2. Calculate the intermediary velocity field $\tilde{\mathbf{u}}(\mathbf{x}, t)$ from

$$\frac{\partial \tilde{u}}{\partial t} = -\frac{\partial u^2}{\partial x} - \frac{\partial uv}{\partial y} - \frac{\partial \tilde{p}}{\partial x} + \frac{1}{Re}\nabla^2 u + \frac{1}{Fr^2} g_x \tag{1}$$

$$\frac{\partial \tilde{v}}{\partial t} = -\frac{\partial uv}{\partial x} - \frac{\partial v^2}{\partial y} - \frac{\partial \tilde{p}}{\partial y} + \frac{1}{Re}\nabla^2 v + \frac{1}{Fr^2} g_y \tag{2}$$

3. Solve the Poisson's equation

$$\nabla^2 \psi = \nabla \cdot \tilde{\mathbf{u}} \qquad in \quad \Omega$$

4. Update the velocities and pressure from:

$$\mathbf{u}(\mathbf{x}, t) = \tilde{\mathbf{u}}(\mathbf{x}, t) - \nabla \psi(\mathbf{x}, t)$$

$$p(\mathbf{x}, t) = \tilde{p}(\mathbf{x}, t) + \psi(\mathbf{x}, t)/\Delta t$$

5. Update the particles position by solving the ODE's:

$$\frac{dx}{dt} = u \qquad \frac{dy}{dt} = v. \tag{3}$$

A number of boundary conditions can be applied at the rigid boundaries, namely: no-slip, free-slip, prescribed inflow, prescribed outflow and continuative flow. At the free surface the usual free surface stress free conditions apply:

$$p - \frac{2}{Re}\left[n_x n_x \frac{\partial u}{\partial x} + n_x n_y \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) + [n_y n_y \frac{\partial v}{\partial y}\right] = 0$$

$$n_x m_y \frac{\partial u}{\partial x} + n_x m_y \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) + [n_y m_y \frac{\partial v}{\partial y} = 0$$

where $\mathbf{n} = (n_x, n_y)$ is the outward unit normal vector to the surface and $\mathbf{m} = (m_x, m_y)$ is the tangential vector. For the Poisson equation the boundary conditions are of mixed type, at the free surface a homogeneous Dirichlet boundary condition applies and at the rigid boundary the boundary condition is of Neuman type.

The implementation of the SMAC method effected in GENSMAC constitutes then of three main parts:

1. The velocity solver based on explicit finite differences

$$\frac{\tilde{u}^{n+1}_{i+\frac{1}{2},j} - u^n_{i+\frac{1}{2},j}}{\Delta t} = \frac{\tilde{p}_{i,j} - \tilde{p}_{i+1,j}}{\Delta x} + \frac{u^n_{i+\frac{1}{2},j-\frac{1}{2}} v^n_{i+\frac{1}{2},j-\frac{1}{2}} - u^n_{i+\frac{1}{2},j+\frac{1}{2}} v^n_{i+\frac{1}{2},j+\frac{1}{2}}}{\Delta y}$$
$$+ \frac{u^n_{i+\frac{1}{2},j} u^n_{i-\frac{1}{2},j} - u^n_{i+\frac{3}{2},j} u^n_{i-\frac{1}{2},j}}{\Delta x} + \frac{1}{Re}\left[\frac{u^n_{i+\frac{3}{2},j} - 2u^n_{i+\frac{1}{2},j} + u^n_{i-\frac{1}{2},j}}{\Delta^2 x}\right.$$
$$\left. + \frac{u^n_{i,j+\frac{3}{2}} - 2u^n_{i,j+\frac{1}{2}} + u^n_{i,j-\frac{1}{2}}}{\Delta^2 y}\right] + \frac{1}{Fr^2} g_x \qquad (4)$$

$$\frac{\tilde{v}^{n+1}_{i+\frac{1}{2},j} - v^n_{i+\frac{1}{2},j}}{\Delta t} = \frac{\tilde{p}_{i,j} - \tilde{p}_{i,j+1}}{\Delta y} + \frac{u^n_{i-\frac{1}{2},j+\frac{1}{2}} v^n_{i-\frac{1}{2},j+\frac{1}{2}} - u^n_{i+\frac{1}{2},j+\frac{1}{2}} v^n_{i+\frac{1}{2},j+\frac{1}{2}}}{\Delta x}$$
$$+ \frac{v^n_{i,j+\frac{1}{2}} v^n_{i,j-\frac{1}{2}} - v^n_{i,j+\frac{3}{2}} v^n_{i,j+\frac{1}{2}}}{\Delta y} + \frac{1}{Re}\left[\frac{v^n_{i+\frac{3}{2},j} - 2v^n_{i+\frac{1}{2},j} + v^n_{i-\frac{1}{2},j}}{\Delta^2 x}\right.$$
$$\left. + \frac{v^n_{i,j+\frac{3}{2}} - 2v^n_{i,j+\frac{1}{2}} + v^n_{i,j-\frac{1}{2}}}{\Delta^2 y}\right] + \frac{1}{Fr^2} g_y \qquad (5)$$

2. The Poisson Solver Based on 5-point FD

$$4\psi_{i,j} - \psi_{i+1,j} - \psi_{i-1,j} - \psi_{i,j+1} - \psi_{i,j-1} = -h^2 D_{i,j}$$

3. Particle Movement

$$x^{n+1} = x^n + u\delta t$$

$$y^{n+1} = y^n + v\delta t$$

3. Parallel Solver

At a given time t the solution domain looks like the one in figure 1 below, where S denotes the cells forming the free surface, B denotes the rigid boundary and F denotes the fluid. The algorithm described in steps 1, 2 and 3 above is to be applied to the cells marked with F only. For the S cells the free surface boundary conditions apply.

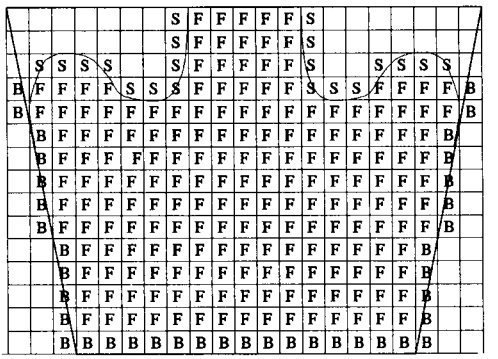


Figure 1: Domain Ω and types of Cells in GENSMAC

The region containing the F cells is split into a number of vertical panels in such a way that each of the panels contains approximately the same number of unknowns. These panels are then assigned to each of the available processors, which goes on to apply the velocity solver to its grid points, see figure 2. For the explicit finite differences scheme this step presents no difficulties for the parallel algorithm, but as mentioned in the introduction, for some problems, stability constraints make the time step extremely small, so that an implicit method is in order. One of the methods we have tried is the use of an implicit discretization for the laplacian operator in the right-hand side of (1-2). Then, to avoid the solution of a 5 diagonal linear system an operator splitting is used to reduce it to the solution of 2 tri-diagonal systems. The resulting scheme is the unconditionally stable Douglas-Rachford scheme (Deville [1974]).

The implicit discrete equations equivalent to equations (3-4) are then: (Deville [1974]).

$$\left(1 - \frac{\Delta t}{Re\Delta^2 x}\right) u^*_{i+1/2,j} = \left(1 + \frac{\Delta t}{Re\Delta^2 y}\delta_y^2\right) u^n_{i+1/2,j} + F(u^n, v^n, \tilde{p}^n) \tag{6}$$

$$\left(1 - \frac{\Delta t}{Re\Delta^2 y}\right) \tilde{u}^{n+1}_{i+1/2,j} = u^*_{i+1/2,j} - \frac{\Delta t}{Re\Delta^2 y}\delta_y^2 u^n_{i+1/2,j} \tag{7}$$

where

$$\delta_x^2 u_{ij} = u_{i+1,j} - 2u_{ij} + u_{i-1j}$$

$$\delta_y^2 u_{ij} = u_{i,j+1} - 2u_{ij} + u_{i,j-1}$$

and $F(u^n, v^n, \tilde{p}^n)$ represents the right hand-side of equation (4) apart from the Laplacian which is being solved implicitly.

The same discretization is used for the v momentum equation (2).

The linear system resulting from (??) can be solved by Gaussian elimination, in each processor separately with no communication, this is due to the fact that the domain is split into vertical strips, producing a matrix in each processor with the first element in the lower diagonal and the last in the upper diagonal equal to zero. This permits the solution of these systems in each processor independently. The linear system (??) has to be solved across processors and an iterative method is better suited for this task, because it requires next neighbour communication only. The matrix being diagonally dominant causes no convergence problems. Figure 2 shows the stencil for the momentum equations.

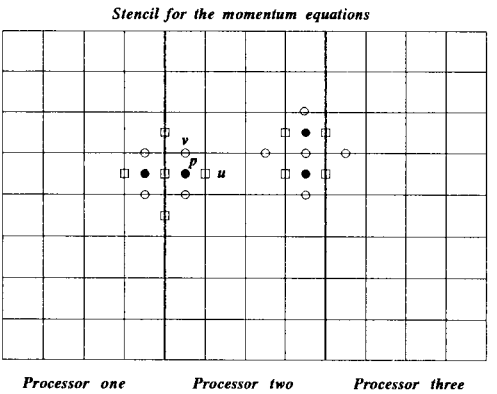


Figure 2

The linear system arising from the 5-point finite differences discretization of Poisson's equation is solved by conjugate gradients. The same data partition used for the velocities solver is used. One step of the conjugate gradient solver requires the computation of the matrix-vector product $y = Ax$, where for a grid point (i, j) the corresponding $y_{i,j}$ is calculated from

$$y_{ij} = \alpha_{ij} x_{ij} + \beta_{ij} x_{i+1,j} + \gamma_{ij} x_{i-1,j} + \delta_{ij} x_{i,j+1} + \epsilon_{ij} x_{i,j-1} \tag{8}$$

and $\alpha, \beta, \gamma, \delta$ and ϵ are the coefficients corresponding to the center, north, south, east and west locations, respectively. These coefficients are derived from the PDE and boundary conditions.

The main difficulty in performing this matrix-vector product is associated with the domain being irregular. This is done in such a way that only points falling inside the region Ω will be included in the calculations. To achieve this, indirect addressing has to be used in order to cope with the contribution of the farthest points (east and west in our case because we are numbering the unknowns by columns). For points inside Ω with neighbours not in Ω, the corresponding coefficient is set to zero. To implement the

correct boundary conditions it is necessary at this stage to modify the coefficients and/or the right-hand side vector as appropriate. Note that the coefficients are formed only once, and are kept fixed during the iterations of the iterative method. This arrangement permits the use of non-uniform meshes. The next step is the solution of the resulting linear system by an iterative method. We make use of the package PIM (Cunha & Hopkins [1993]) for the iterative solver and global communication routines. PIM requires the user to provide the matrix-product routines. These routines implement formulae (??) taking into consideration the fact that our region is arbitrary. The generality of the geometry makes the next neighbour communication pattern much more involved than the case of a square region, because the length and positions of the vectors to be exchanged depend now on the geometry. Figure 2 below presents the performance of the Poisson solver when applied to a typical problem with increasing number of mesh-points.

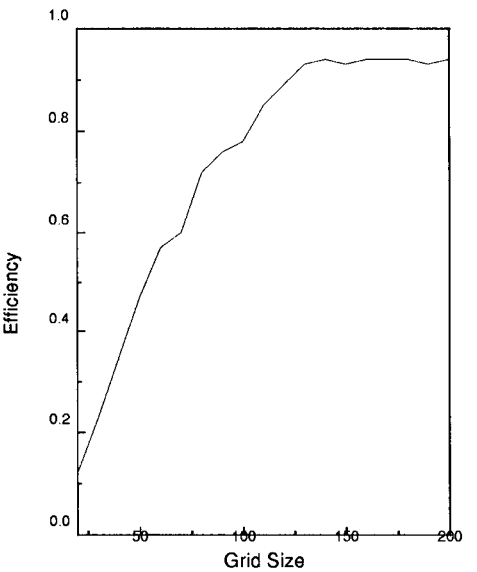


Figure 3 $E = T_s/(pT_p)$, T_s is the time for the sequential code and T_p is the time for the parallel code using PVM with 4 processors.

The parallel implementation of the particle movement is done likewise. Each processor having upgraded the velocity and pressure fields applies the ODE (3) to find the position of its particles at that time. Particle movement across processor boundaries will be necessary and will only involve next neighbour communication. The position of the fluid is updated with each processor communicating the possible particles which are to be moved across the boundaries. At the end of a computing cycle all processors hold the correct information for performing the next step.

Load Balance

The load balance presents a major difficulty for problems with free surfaces, because the domain changes as time evolves. This is specially so for the type of problems we are dealing with in this work, as the free surface can take arbitrary geometries and can change shape dramatically with time. For instance, in a problem with inflow, in the initial phase the region occupied by the fluid is small, and so is the computation involved, so that it is not worth distributing it. As the domain grows more work needs to be performed and it is then distributed amongst the available processors. The strategy we adopted in the parallel version of GENSMAC is a follows:

1. We start running the code in a single processor. As the number of cells grows sufficiently large, the domain is split into two and the work distributed accordingly.

2. After a fixed number of time steps each processor reports to the root processor its cell count. The root processor adds up this incoming information and decides when it is time to add a new processor and redistribute the work. This is repeated until all processors are used up.

The above strategy works very well in balancing the work load but it inserts bottlenecks and synchronization points into the code. It also increases the communication.

Another technique we have tried is to divide the whole domain into strips of equal width at the beginning of the computation and assign each of these strips to one processor, which will remain fixed throughout the computation. Processors with part of the domain containing no fluid remain idle until fluid arrives at those parts. This simplifies the coding substantially and also saves on communication. Despite the fact that some processors remain idle for some time, for the problems we have been running our code on we cannot see much difference in performance.

This work is part of an on-going project supported by FAPESP under grant #93/3622-7. The code is implemented in FORTRAN 77 using PVM for message passing.

REFERENCES

1. Amsden, A.A.; Harlow, F.H. - The SMAC method: A numerical technique for calculating incompressible fluid flows - Los Alamos Scientific Laboratory, Report LA-4370 - [1971]
2. Cunha, R.D. - A Study of Iterative Methods for the Solution of Systems of Linear Equations on Transputer Networks - Ph. D. Thesis - University of Kent at Canterbury U.K. - [1992]
3. Cunha, R.D.; Hopkins, T. - The parallel solution of linear equations using iterative methods on transputer networks - Technical Report 16/92 - University of Kent at Canterbury U.K. - [1992a]
4. Cunha, R.D.; Hopkins, T. - PIM 1.1 - The parallel iterative package for systems of linear equation - User's guide - Technical Report - University of Kent at Canterbury U.K. - [1993]
5. Deville, M. O. - Numerical Experiments on the MAC code for a slow flow. J. Comp. Physics - 15, pp. 362-374 [1974]

6. Harlow, F.; Welsh, J.E. - Numerical calculation of time-dependent viscous incompressible flow of fluid with free surfaces - Phys. Fluids **8** pp. 2182-2189 [1965]
7. Tome, M.F.; McKee, S. - GENSMAC: A computational marker and cell method for free surface flows in general domains - J. Comp. Physics **110** pp. 171-186 [1994]
8. Tome, M. F. - GENSMAC: A Multiple Free Surface Fluid Flow Solver - Ph. D. Thesis University of Strathclyde - Glasgow - Scotland [1993]

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A new domain decomposition method using virtual sub-domains

Akio Ochi, Yoshiaki Nakamura and Hiromitsu Kawazoe
Department of Aerospace Engineering, Nagoya University, Chikusa-ku,
Nagoya 464-01, Japan

The domain decomposition technique is one of key issues in parallel computation. Especially, when we employ unstructured grid systems, it influences the efficiency of parallel processing. In this paper, a new domain decomposition method using virtual sub-domains and a genetic algorithm is described. Furthermore, this new method has a capability to perform partitioning grid in parallel processing. The new method was applied to the flow around a cylinder to examine its performance under a workstation cluster.

1. INTRODUCTION

The unstructured grid system has been widely used in CFD, because of its advantages over the structured grid to treat complicated geometries and to automatically generate grids. However, its parallelization needs some extra work compared with that in structured grids. In parallel CFD, domain decomposition is mostly employed to distribute the work load of computation to each processor. That is, while the domain decomposition in structured grids is simple and can be performed automatically by a parallel compiler, it is more complicated in unstructured grids. Because in two dimensional structured grids we can easily divide the whole grid by grid lines with i=constant or j=constant. However, there are no such lines in unstructured grid systems. In the present paper, a domain decomposition method for unstructured grids is discussed.

There are several standard domain decomposition methods proposed such as the graph bisection method, the coordinate bisection method, and the spectral bisection method [1]. They have both pros and cons. Though the graph bisection method and the coordinate bisection method require small computational costs, the quality of partitioning is not good. On the other hand the spectral bisection method shows good results, but it is expensive in computational cost. Because we need to calculate the eigenvalues and the eigenvector of an NxN matrix, where N is the number of grid points. Furthermore, these three methods divide a region not into arbitrary number of sub-regions, but into halves. Because they are based on the bisection method. The bisection method produces more than three regions using it recursively. Hence, they are not able to make the globally optimal decomposition.

In this study, we developed a new domain decomposition method for unstructured grids. The outline of the new method is shown in fig. 1. Suppose that there is an unstructured grid shown in fig. 1a. In the first step, the whole mesh is divided into small sub-domains (fig. 1b). We call these small sub-domains

virtual sub-domains. In fig. 1b, there are one hundred virtual sub-domains. Then allocate each virtual sub-domain to node processors by using the genetic algorithm so that the high quality partitioning is obtained (fig. 1c). The high quality means both a minimum of communication cost and a good load balancing when solving the Navier-Stokes equations or the Euler equations on parallel machines. Finally, the grid elements in each sub-domain are allotted to each processor (fig. 1d). In the present study one hundred virtual sub-domains were allotted to six processors.

2. VIRTUAL SUB-DOMAIN

Virtual sub-domains are small domains in the whole region, where there are tens to hundreds sub-domains. The reason why we call such small domains the "virtual" sub-domains is that we are not aware of these small domains after grid partitioning is completed. The virtual sub-domains are utilized only in the procedure of partitioning.

The way how to make virtual sub-domains is not a key issue in the new decomposition method. In this study we employed a method shown in fig. 2, by which we can make the virtual sub-domains quickly and with a small computational cost. Figure 2a shows an unstructured grid to calculate the supersonic flow around a cylinder and a rectangular frame containing the whole grid. This frame is a starting domain. Then the longer side of this frame is divided into halves. As a result we have two domains (fig. 2b). To make the subsequent sub-domains, we count the number of grid cells included in each sub-domain. Here, we suppose that the upper sub-domain has more grid cells. Therefore the upper sub-domain is divided with regard to the longer side (fig. 2c). In fig. 2c the lower sub-domain has the largest number of grid cells, hence this sub-domain will be divided. Thus we have four sub-domains as shown in fig. 3d. This procedure will be repeated until the number of sub-domains reaches a specified number.

A major advantage of using these virtual sub-domains is in that the cost for decomposition does not increase with the number of grid points. Because the partitioning cost depends on the number of virtual sub-domains in our new method, while those in other methods depend on the number of grid points. The work load for making virtual sub-domains is proportional to the number of grid points, though the work load for making virtual sub-domains accounts for a very small percentage of the whole partitioning cost (approx. less than 3%).

3. GENETIC ALGORITHMS

We utilized the genetic algorithms to distribute virtual sub-domains into node processors. The genetic algorithms were developed in 1970s by John Holland et al. [2] to simulate some processes in the evolution problem in nature. They are efficient and robust algorithms to solve complex optimizing problems. In the genetic algorithms the solution of a problem is represented by chromosomes, where many chromosomes are initialized at random.

Chromosomes make their children, who in turn give birth to grandchildren of chromosomes. After tens to hundreds generations, chromosomes will evolve to a nearly optimal solution.

In the present study, fifty chromosomes were employed, and fifty to one hundred generations were repeated. Generally, more chromosomes and more generations are used in the conventional genetic algorithms. However, we could successfully reduce them by two hybrid operators proposed in this study.

Outline of the genetic algorithms

i) Initialize a group of chromosomes.

ii) Evaluate and rank chromosomes.

iii) Crossbreed chromosomes to produce new chromosomes, where some operators are performed to modify chromosomes.

iv) Remove lower rank chromosomes from the group to secure the space for new chromosomes.

v) Evaluate and rank new chromosomes, and then put higher rank chromosomes into the group.

vi) If the score is converged, the output is the best chromosome. Otherwise, get back to step iii) and repeat it.

3.1. Evaluation function

The evaluation function is used to score chromosomes. The chromosome with a higher score has a larger probability of breeding child chromosome than that with a lower score. Since the evaluation function considerably influences both the quality of solution and the convergence rate, we should determine the evaluation function very carefully. Here in this study the following function Φ was employed.

$$\Phi = \max_{i=1,N_{proc}} \Phi_i, \quad \Phi_i = \frac{N_{element,i}}{A_i} + \frac{N_{trans,i}}{B_i}$$

where

N_{proc} : number of processors
$N_{element,i}$: number of grid elements assigned to the i-th processor
$N_{trans,i}$: number of points with which the i-th processor will communicate
A_i : calculation speed of the i-th processor [elements/sec]
B_i : communication speed of the i-th processor [points/sec]

The total evaluation function Φ is a maximum of the evaluation function Φ_i at each processor. Φ_i represents a time required to advance by one time step, which consists of two terms; the first term denotes a pure calculation time, and the second term a communication time. Hence our objective with regard to mesh partitioning is to decrease the value of Φ. Thus a chromosome with a smaller value of Φ has a higher score. Since we can give different values for A_i and B_i, the new partitioning method can be applied to the system of heterogeneous parallel machines which have different types of CPU and network structure.

3.2. Operators

Operators were utilized to emulate mutations that are common processes in nature to change chromosomes at crossbreeding. Actually in the present study the following four operators were used: i)mutation operator, ii)two points crossover operator, iii)hybrid operator #1, iv)hybrid operator #2 (see fig. 3). The first and second operators are often used in the conventional genetic algorithms. The third and fourth operators are developed in this study especially for the domain decomposition.

3.3. Parallelization of the genetic algorithms

One of the reasons why we employed the genetic algorithms was that it has a capability to parallelize the domain decomposition. The genetic algorithms are easy to apply to parallel processing. Because chromosomes evolve in each processor in the same fashion as life evolves in an isolated island. At every ten generations, the best chromosome in each processor is sent to a host processor, and then the best chromosome among the whole chromosomes is distributed to all the processors to accelerate the evolution. However we are afraid that this technique might cause a local minimum solution in convergence. Therefore we should be careful to use this technique in this respect.

4. WORKSTATION CLUSTER

A workstation cluster shown in fig. 4 was used to perform parallel computation. Each machine has a different performance, which is connected to each other with the Ethernet (10Mbps).

The PVM was employed to exchange data, which is a software package that allows a heterogeneous network of parallel and serial computers to appear as a single concurrent computational resource. The PVM was developed in 1989 at Oak Ridge National Laboratory (ORNL), and the version of 3.3.6 is employed in the present study.

5. RESULTS AND DISCUSSION

Figure 5 shows a typical change of decomposition, where the best chromosomes at each generation are shown. The left hand side picture shows virtual sub-domains allotted to six processors, and the right hand side figure shows the quality of partitioning, where the vertical axis of the figure is an evaluation function Φ_i. The distributions are given at random in the first generation. Then, they are refined with generation. Consequently the communication cost is reduced and the load balancing is improved.

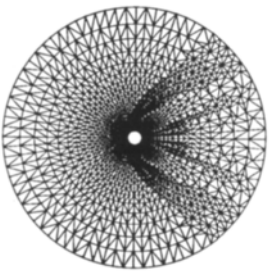
The new method was applied to two types of grid system to examine its performance. The first one is a supersonic flow around a cylinder (Type A), and the second is an internal flow inside transformer (Type B). The number of grid points is 9056 (Type A), and 36364 (Type B). Both grids are shown in fig. 6. The results are compared with the coordinate bisection method in figs. 7 and 8, which show partitioned grid elements and the quality of partitioning. The coordinate bisection method can produce a good load-balanced partitioning,

while its communication cost is not optimized. The graph bisection method shows the same tendency as the coordinate bisection method, and it has slightly poor results than the coordinate bisection method, though it is not shown in this figure. On the other hand, the new method could not produce a load balanced partitioning. However the communication cost was reduced compared with the coordinate bisection method. Thus, the new method shows slightly better results than the coordinate bisection method.

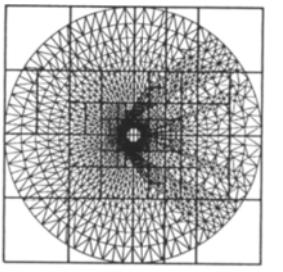
As regards the partitioning speed, the new method is inferior to the coordinate bisection method for this grid size, though parallel processing can be applied to it. However, the difference between the new method and the coordinate bisection method will be reduced with increasing the number of grid points. It seems that both methods will show similar performances for nearly 100,000 grid points.

6. CONCLUSIONS

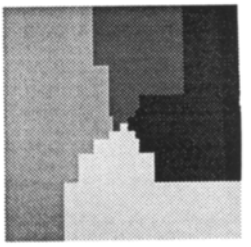
The new method proposed in this paper did not show so good a performance as expected for sample problems described in this paper. However it is expected that it would show a better result for a larger gird system. Furthermore it will be possible to improve this method, particularly the operators used in the genetic algorithms. In addition, the evaluation function is also so important that further study is required.



a) Grid in the whole region.



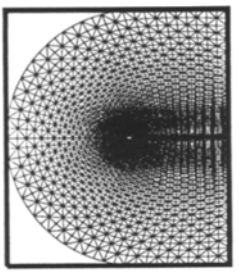
b) The whole region divided into 100 virtual sub-domains.



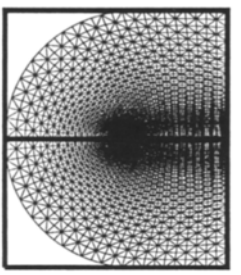
c) 100 virtual sub-domains allotted to 4 node processors by the genetic algorithms.

d) Partitioned grid elements.

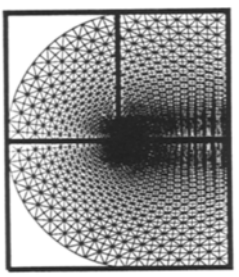
Fig. 1 Overview of the new domain decomposition method.



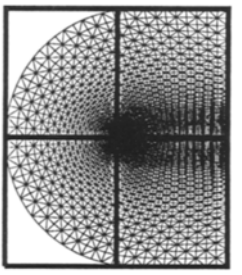
a) Unstructured grid and a frame enclosing the grid.



b) Two sub-domains (upper and lower).

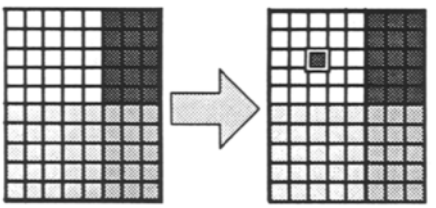


c) Three sub-domains

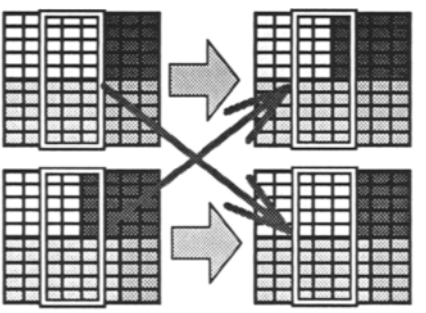


d) Four sub-domains.

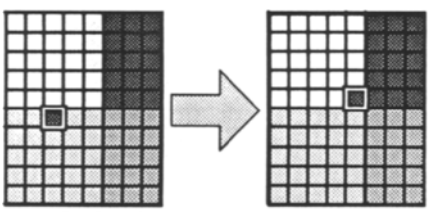
Fig. 2 Procedure of dividing the whole region into virtual sub-domains.



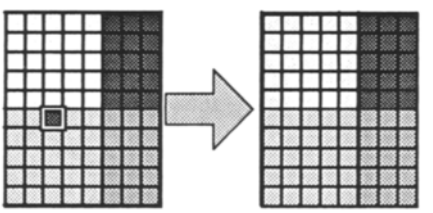
a) Mutation operator.
Change allocation at random.



b) Cross over operator
Swap a group of sub-domains.



c) Hybrid operator #1
Move an isolated sub-domain.



d) Hybrid operator #2
Change an isolated sub-domain.

Fig. 3 Operators for the genetic algorithms.

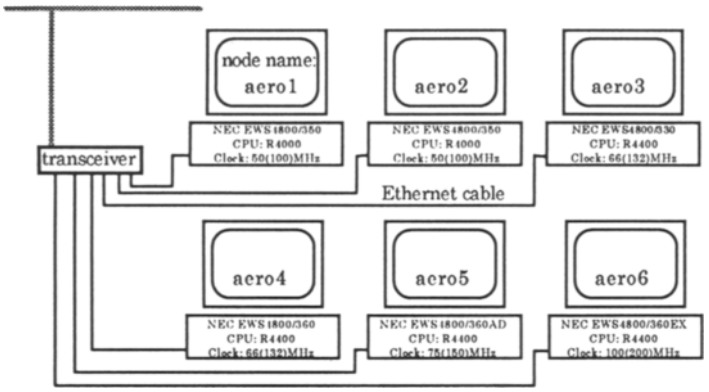


Fig. 4 Workstation cluster.

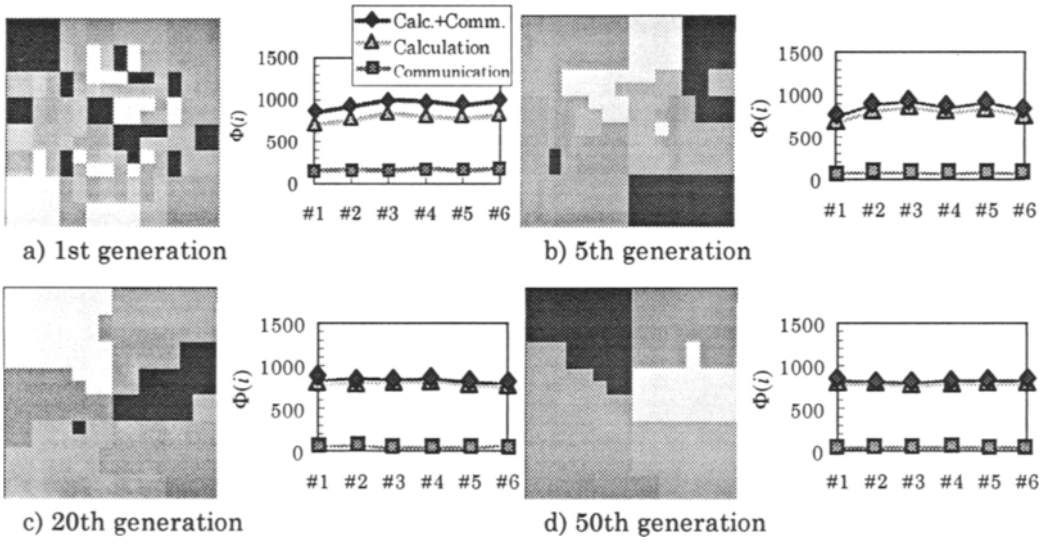
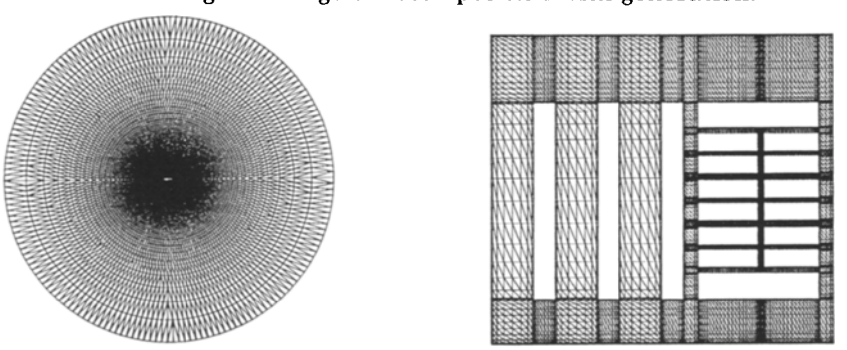
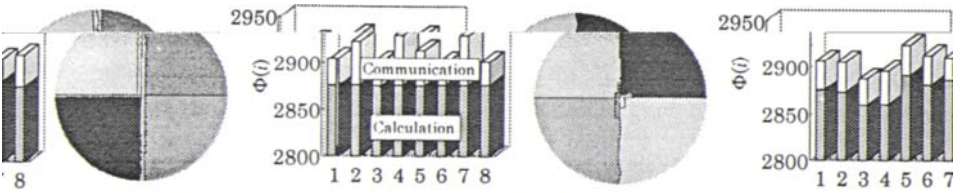


Fig. 5 Change of decomposition with generation.



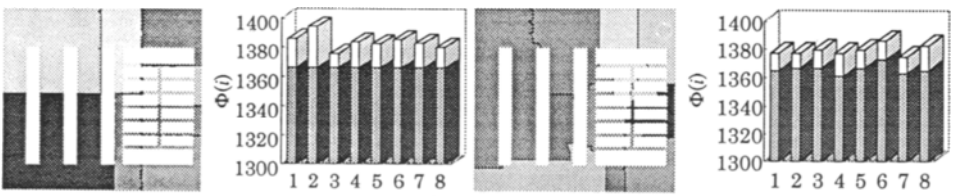
a) Type A grid (cylinder). b) Type B grid (transformer).

Fig. 6 Two types of grid system to verify the new method.



a) Coordinate bisection method. b) New method

Fig. 7 Decomposition of type A grid, calculation and communication costs are shown as a bar chart.



a) Coordinate bisection method. b) New method

Fig. 8 Decomposition of type B grid, calculation and communication costs are shown as a bar chart.

REFERENCES

[1] Barth, T.J.: Unstructured Grid Methods for Advection Dominated Flows, AGARD Report 787, 6-1-6.61 (1991).

[2] Holland, John., *Adaptation in Natural and Artificial Systems*, Ann Arbor: University of Michigan Press, (1975).

[3] Lawrence Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, (1990).

[4] A. Geist, A. Begulin, J. Dongrra, W. Jiang, R. Manchek, and V. Sunderam, "PVM 3 User's Guide and Reference Manual", May 1993.

[5] Hirsch, C.,"Numerical Computation of Internal and External Flows", Vol.2,John wiley and Sons, Inc, New York. (1988),pp.493-556.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A commercial CFD application on a shared memory multiprocessor using MPI

Paul F. Fischer[a] and Madan Venugopal[b]

[a]Department of Applied Mathematics, Brown University, Box F
Providence, RI 02912 USA

[b]Supercomputing Applications, Silicon Graphics Inc., MS 580
2011 N. Shoreline Blvd., Mountain View, CA 95014 USA

The implementation of a commercial spectral element CFD code on a shared memory parallel computer using the recently defined Message Passing Interface (MPI) standard is discussed. The application was originally implemented in parallel using the NX message passing library; issues regarding the implementation and performance of the MPI-based version in a shared memory environment are discussed. In addition, recent algorithmic improvements which reduce the solution time for unsteady flow calculations are presented. Performance results are presented for the analyses of two and three dimensional flow problems on up to eight processors. Modifications to the parallel implementation which will improve the computation to communication ratio through reduced latency are suggested.

1. Introduction

Existing computational fluid dynamics applications (CFD) have been predominantly based on vector algorithms simply because the fastest machines have been vector supercomputers till recently. However, with the availability of relatively inexpensive RISC microprocessor based multiprocessor machines the definition of supercomputing is changing. In line with this trend has been the development of parallel CFD algorithms with a view to achieving possible order of magnitude speedups using a large number of processors. A significant advance in this regard has been the development of the parallel spectral element algorithm [1] for incompressible flow problems.

The current paper discusses the implementation of Nekton, a parallel spectral element application, on the Silicon Graphics(R) Power Challenge(TM) shared memory parallel computer. Nekton(TM) was originally implemented using the NX message passing library. The current implementation on the Power Challenge uses the recently defined Message Passing Interface (MPI) standard [2]. Nekton and the Power Challenge system will be briefly described, followed by a discussion of the implementation of Nekton on a Power Challenge using MPI. The implementation of an improved pressure solver in Nekton which reduces the computation time significantly is discussed. Finally, results are presented from the analyses of some interesting problems in fluid dynamics using Nekton on the Power Challenge system.

2. Nekton

Nekton is a spectral element code for modelling two dimensional and three dimensional incompressible fluid flow problems. The fluid domain is discretized into macro (spectral) elements in an unstructured grid. Within each element, the solution variables and geometry are represented as Nth-order tensor product polynomial expansions (typically $N \sim$ 4-12), which allow for locally structured (matrix-matrix product based) data accesses during operator evaluation. Nekton employs iterative solvers based on preconditioned conjugate gradients which offer savings in memory requirements and processing time. The spectral element algorithm makes use of the native parallelism existing in the problem by mapping groups of spectral elements to separate processors according to standard SPMD message passing paradigms. The locally-structured/globally-unstructured nature of the spectral element method enables an efficient parallel implementation. One of the main advantages of the spectral element algorithm is the very fast convergence rate per degree of freedom. Solutions in moderately complex geometries can be represented with a few elements and fast time to solution can be achieved in the modelling.

Parallelism is implemented in Nekton in a general message passing scheme. Since only the residual updates (nearest neighbor exchanges) and inner products require communication the machine dependent part of the code restricted to a narrow region. As a result, the same code can be used for uniprocessor machines as well as parallel machines. This feature made the porting of the application to the SGI Power Challenge system using the MPI message passing library relatively easy. Details of the implementation are given below.

3. The Silicon Graphics Power Challenge System

The Silicon Graphics Power Challenge system is a shared memory parallel computer which can have upto 18 75 MHz R8000(TM) processors. The R8000 processor is a 64-bit superscalar superpipelined RISC microprocessor which can execute four floating point operations per cycle to give a theoretical peak performance of 300 Mflops.

The Power Challenge architecture is shown in figure 1. The 18 processors are configured on 9 boards with each processor having access to 4MB of 2-way interleaved 4-way associative streaming cache. The cache is connected to main memory via a system bus with a bandwidth of 1.2 GB/sec. The main memory can consist of up to 8 boards each with a maximum of 1 GB of 2 way interleaved memory.

The IRIX(TM) 6.0 software on the Power Challenge provides a 64-bit UNIX(TM) operating system and 64-bit compilers.

4. Message Passing Interface

The Message Passing Interface (MPI) is a standard recently defined by group of computer scientists , hardware vendors and applications scientists interested in developing a uniform interface, for message passing applications, which was relatively machine independent. [2]. The standard was finalized quite recently (May 1994). However, there are

Power Challenge System Block Diagram

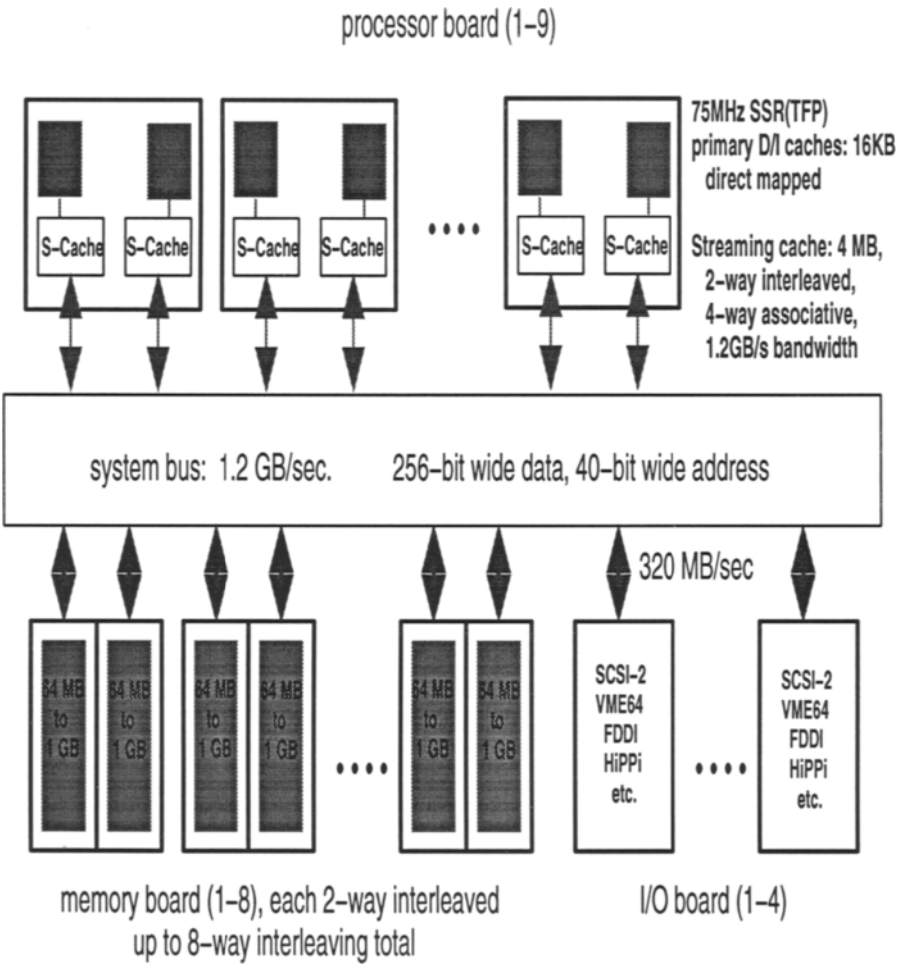


Figure 1

already several hardware implementations of MPI. The implementation used in Nekton on the Power Challenge is `mpich`, which was developed at Argonne National Laboratory [3,4], and is based on the chameleon and p4 message passing libraries [5–7].

The current Power Challenge implementation of MPI uses shared memory for communication within the system. However, it can also communicate with other machines on a TCP/IP network using sockets. Although the underlying mechanism for the message passing is transparent to the user, the actual implementation determines the performance in terms of the bandwidth and latency. The number of processes can be controlled at run time via a procgroup file. The contents of this file determine how many processes are spawned and whether these are on the local machine or on a remote machine. If p processes are defined and there are p or more processors on the system, the IRIX operating system maps these processes onto the processors. The processes then communicate with each other using shared memory regions accessed via one of the protocols described below.

The MPI implementation on the Power Challenge was relatively straightforward considering that it was one of the first 64 bit implementations of MPI. The initial implementation of MPI on the Power Challenge using UNIX System V shared memory interprocess communication (IPC) gave a latency of about 1500 microseconds. Implementing the shared memory communication using SGI IRIX IPC with semaphores and shared memory caused the latency to reduce to about 400 microseconds. Using spin locks instead of the semaphores caused the latency to drop even further to about 90 microseconds. The changes to implement these variations of shared memory access were restricted to a narrow region of the `mpich` code.

The IRIX IPC and spin lock based implementation requires a shared region in memory called an arena to be defined and initialized. This is implemented via a shared arena file which is mapped into each process's user space. Writing in this space is controlled by busy-wait locks for synchronization called spin locks, while all processes can read freely from this space. Messages to be sent are queued in the shared arena buffer and messages received are dequeued from this buffer. All processes using the same arena have access to the same set of IPC mechanisms, resulting in a lightweight software access to the spin locks without the need for explicit (expensive) system calls.

5. Implementation in Nekton

Since the communication routines within Nekton are based on a small set of basic functions, the actual implementation of Nekton on Power Challenge consisted mainly of building the appropriate interface routines for MPI as well as modifying the build structure to incorporate the initialization, I/O and cleanup required by MPI. The NX calls were thus replaced by the equivalent MPI calls [4].

The single processor performance was improved by hand tuning the small matrix multiply kernels in Nekton. The primary technique used was explicit loop unrolling, which resulted in performance within 50% of assembly coded performance.

6. Projection Methods

A recently developed projection technique [8] was implemented in order to reduce the pressure iteration count for time dependent problems. The method is based on the observation that the pressure typically changes very little from one time step to the next, implying that previous solutions form a good basis for approximating the solution at the current step. If the approximation is not sufficiently accurate, further conjugate gradient iterations can be taken to reduce the residual of the remaining perturbation to a desired tolerance. This scheme can be implemented in an essentially black-box fashion, provided one has calling access to the system solver and to the forward operator evaluation.

The method is developed as follows. Let $A\underline{x}^n = \underline{b}^n$ represent the $m \times m$ symmetric positive definite system governing the pressure at time level n, with $\underline{x}^n$ the vector of nodal unknowns for pressure and $\underline{b}^n$ the corresponding right hand side vector. Both vectors are assumed to be distributed according to the spectral element-to-processor map. Let $X_l = \{\tilde{\underline{x}}_1\, \tilde{\underline{x}}_2 \ldots \tilde{\underline{x}}_l\}$ be the $m \times l$ matrix having columns which are distributed in the same fashion as $\underline{x}^n$ and which satisfy the following properties:

$$\tilde{\underline{x}}_i \in span\{\underline{x}^{n-1}, \underline{x}^{n-2}, \ldots, \underline{x}^{n-l}\} \tag{1}$$

$$\tilde{\underline{x}}_i^T A \tilde{\underline{x}}_j = \delta_{ij} \tag{2}$$

where δ_{ij} is the Kronecker delta. The best fit approximation with respect to the A-norm ($||\underline{x}||_A \equiv (\underline{x}^T A \underline{x})^{\frac{1}{2}}$) is given as $\bar{\underline{x}} = X_l X_l^T \underline{b}^n$, which can be computed at the cost of l simultaneous inner products followed by l fully parallel SAXPY operations. The entire projection algorithm is given as

- compute $\bar{\underline{x}} = X_l X_l^T \underline{b}^n$
- solve $A\tilde{\underline{x}} = \underline{b}^n - A\bar{\underline{x}}$ to tolerance ϵ
- set $\underline{x}^n = \bar{\underline{x}} + \tilde{\underline{x}}$
- update $X_{l+1} \longleftarrow X_l$: $\quad \tilde{\underline{x}}_{l+1} = (\tilde{\underline{x}} - X_l X_l^T A\tilde{\underline{x}})/||\tilde{\underline{x}} - X_l X_l^T A\tilde{\underline{x}}||_A$

Per time step, the algorithm requires $2 \times l$ inner-products of length m, $2 \times l$ SAXPY's, and two applications of the sparse operator A. When l exceeds a prescribed value (typically $l_{\max} \sim 20$), we restart the basis set with $X_1 = \{\tilde{\underline{x}}_1\}$ satisfying (1-2).

7. Results from Analysis

Nekton implemented as above on the Power Challenge was used to model the following two and three dimensional incompressible fluid flow problems. Results are presented in the attached tables.

7.1. Cylinders in a duct

This is a three dimensional steady Stokes flow problem in a square duct with two transversely mounted cylinders, following [1]. In the current implementation it is modelled using 32 spectral elements of order 9. Power Challenge results are presented in Table 1 along with comparisons with the Cray(R) X-MP(TM) and iPSC/2-VX/d4(TM) (16 processor) which are derived from [1]. Since a later version of Nekton with an accelerated convergence criterion was used for the Power Challenge runs the present results had to be scaled down to account for the reduced number of computations.

Table 1
Computing times and MFLOPS for three dimensional Stokes flow problem

Machine	Processors	Wall Clock, seconds	Efficiency	Estimated Mflops
SGI Power	1	76	100%	66.2
Challenge	2	40	95%	125.8
..	4	24	79%	209.7
..	8	18	53%	279.6
Cray X-MP	1	–	100%	66.0
iPSC/2-VX/d4	16	130	75%	44.0

Times are wall clock times in seconds

7.2. Oscillating cylinder in a uniform flow

Flow over an oscillating cylinder in a uniform flow is simulated with 186 spectral elements of order 9. The cylinder oscillation is a harmonic function. The implementation of a moving body in the fluid is achieved by a coordinate transformation of the governing equations of the problem as discussed in [9]. The mesh was provided by Copeland [9] and the problem was formulated to compare with experimental results [10]. The Reynolds number for the simulation was 80. The times in Table 2 are for 2401 time steps and represents a typical simulation of about fifty cycles of oscillation at vortex shedding lock-in. Oscillation with different frequencies can require significantly more CPU time and as such it can be very expensive to generate several sets of data with multiple frequency components. Improvements in performance therefore directly increase the potential to explore a larger region of the parameter space for this problem.

Table 2
Computing times for oscillating cylinder problem on the SGI Power Challenge

Number of processors	Wall Clock time
1	9 : 41 : 44
8	3 : 10 : 03

Times are wall clock times in hours:minutes:seconds

With the new pressure solver, the improvement in performance for the same problem is shown in Table 3. Times are for 100 time steps.

8. Discussion of results

The parallel performance and efficiency depend primarily on the single processor performance, ratio of computation to communication for the problem and the latency of the message passing library.

The single processor performance of Nekton is determined by the performance of small matrix multiplies. The size of the matrices depends on the order of the spectral elements chosen for the analysis; the inner product length is always equal to the order of the expansion, plus or minus one. For the SGI Power Challenge implementation these small

Table 3
Projection method timings for 100 steps of the oscillating cylinder problem.

Processors	Standard solver		Projection solver	
	Time, secs	Efficiency, %	Time, secs	Efficiency, %
1	1636	100	1338	100
2	898	91	756	88
4	538	76	372	90
8	470	44	317	53

Times are wall clock times in seconds

matrices fit into secondary cache and the product routine can be optimized to be minimize cache misses.

The computational complexity of the spectral element method has a leading order computation to communication ratio of $O(N^2)$ in either two or three dimensions. However, the larger granularity of 3D problems results in a relatively lower penalty for latency-dominated communications. Inner products are an example of latency dominated communications since only one word needs to be transferred from each processor ($\log p$ times) to compute the inner product of distributed vectors. In addition, the original spectral element formulation in Nekton was optimized for 3D problems - no attempt was made to bundle multi-element data shared by two processors since it was assumed that the message sizes for each element interface (N^2 words) would cover latency effects. In 2D, messages of only size N incur a relatively high latency penalty, and we have recently developed a bundled gather-scatter algorithm which should reduce this penalty. Results will be reported in a future publication.

We underscore the importance of latency by noting that, with the original latency of about 1500 microseconds, Nekton was slower on two processors than on one. With the latency reduced to 400 microsecs we obtained a parallel efficiency of 84% on 2 processors and 38% on 4 processors. The final figures and improved efficiencies with a latency of 90 microseconds are shown in Table I above. The iPSC efficiency of 75% on 16 processors [1] is higher than the Power Challenge efficiency on 8 processors, although the absolute performance in terms of time to solution is significantly slower than even the single processor time on the Power Challenge. In addition to the algorithmic improvements mentioned above, we expect a further reduction in latency will be attained when the public domain software `mpich` is replaced by a native SGI implementation of MPI which is currently under development.

9. Conclusions

This paper presents the implementation of a parallel spectral element CFD application, Nekton, on the SGI Power Challenge, using the newly defined Message Passing Interface standard. This enables a smooth transition of a code designed for a distributed memory system to a shared memory parallel computer. Results from the analysis of 2D and 3D fluid flow problems are presented which show good parallel speedups and efficiencies with increasing number of processors. Factors which limit the speedup and efficiency and

strategies to improve them are discussed.

A novel projection technique for computing approximate solutions to systems of linear equations is implemented in Nekton to give a good initial guess to the iterative solver. The resulting solver shows 20 to 50% improved performance on single and multiple processor simulations of dynamic flows.

Acknowledgements

The authors wish to thank Fluent Inc. for the use of Nekton v 2.9 in running the simulations discussed in this paper. Nekton is a product of Fluent Inc of Lebanon, NH and Nektonics of Cambridge, MA.

REFERENCES

1. Fischer, P.F., Ho, L-W, Karniadakis, G., Ronquist, E.M., Patera, A.T. *Recent advances in parallel spectral element simulation of unsteady incompressible flows* Computers and Structures vol. 30, no. 1/2, pp 217-231, 1988
2. *MPI: A message-passing interface standard* International Journal of Supercomputing Applications, 8(3/4), 1994
3. Gropp, W., Lusk, E., and Skjellum, A, *Using MPI: Portable Parallel Programming with the Message Passing Interface* MIT Press, 1994.
4. Gropp, W., Lusk, E., Doss, N., Skjellum, A. *MPICH Model MPI Implementation Reference Manual Draft* Argonne National Laboratory Report ANL-94/00, Nov. 1994.
5. Bridges, P., Doss, N., Gropp, W., Karrels, E., Lusk, E., and Skjellum, A, *User's Guide to* `mpich`, *a Portable Implementation of MPI* Argonne National Laboratory Report 1994.
6. Butler, R. and Lusk, E., *User's Guide to the p4 Parallel Programming System* ANL Technical Report ANL-92/17 April 1994.
7. Boyle, J., Butler, R., Disz, T., Glickfield, B., Lusk, E., Overbeek, R., Patterson. J, Stevens, R., *Portable Programs for Parallel Processors* Holt Rinehart and Winston Inc. 1987
8. Fischer, P.F., *Projection techniques for iterative solution of* $A\underline{x} = \underline{b}$ *with successive right-hand sides. ICASE Report No. 93-90*, NASA CR-191571.
9. Copeland, S. and Cheng, B. *A numerical investigation of vortex shedding from a transversely oscillating cylinder* 6th Intl Conference on Flow-induced vibration, April 1995
10. Venugopal, M. *Damping and response prediction of flexible cylinders in a sheared flow* Ph.D thesis in Ocean Engineering, Massachusetts Institute of Technology, *in preparation.*

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel Computation With the Spectral Element Method

Hong Ma†

Department of Applied Science, Bldg. 490D
Brookhaven National Laboratory
Upton, NY 11973, U.S.A.

ABSTRACT

Spectral element models for the shallow water equations and the Navier-Stokes equations have been successfully implemented on a data parallel supercomputer, the Connection Machine model CM-5. The nonstaggered grid formulations for both models are described, which are shown to be especially efficient in data parallel computing environment.

1 Introduction

The development of the spectral element algorithms and their applications in solving incompressible flow problems [1, 2, 4, 5, 6] have demonstrated the enormous potential of the p-type of finite element method in improving computational efficiency. Like the spectral method, the spectral element method uses high-order polynomials as trial functions, but like the finite element method, it decomposes the computational domain into many elements and defines local trial functions. The hybrid character of the spectral element method enables it to overcome the shortcomings of both the spectral method and the finite element method but still retains their advantages. Since the trial functions of the spectral element method are local, it can handle complex geometry easily. On the other hand, it still is a high-order weighted residual method, so the exponential convergence rate is achieved as the order of polynomials is increased. Therefore, it is more efficient than low-order numerical methods, such as the finite difference and finite element methods.

Another important aspect which greatly enhances the computational efficiency of the spectral element method is the natural fit of this method to parallel computing. The main difference between the spectral element method and the spectral multi-domain method is that the C^0 and C^1 boundary conditions at the interface of the elements have to be enforced explicitly in the latter. By contrast, the spectral element method uses the variational principle to guarantee C^0 and C^1 (weakly) continuity at the interface,

†This work received support from The U. S. Department of Energy under contract No. DE-AC02-76CH00016, and from The National Science Foundation through grant OCE-9312324. Most of the computation was performed on the CM-5 at ACL of Los Alamos National Laboratory.

therefore, parallel algorithms can be implemented conveniently. Fischer and Patera [1] successfully implemented their spectral element model on MIMD (Multiple Instructions, Multiple Data) type of computers.

This paper is about the development and application of the parallel spectral element models for the Navier-Stokes equations and the shallow water equations with nonstaggered grid formulations. The performance of these models on the Connection Machine system, which is basically a SIMD (Single Instruction, Multiple Data) type of architecture, is analyzed.

2 Governing Equations

In the present work, we study irrotational and rotational flows, which are governed by the following two sets of equations, respectively:

Incompressible Navier-Stokes Equations:

$$\frac{\partial \mathbf{u}}{\partial \mathrm{t}} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla \mathrm{p} + \frac{1}{\mathrm{R_e}} \nabla^2 \mathbf{u} \tag{2.1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2.2}$$

where $\mathbf{u} = (\mathrm{u}, \mathrm{v}, \mathrm{w})$ is the velocity vector (u, v, and w are the velocity components in the x, y, and z directions, respectively); p is the pressure; $R_e = UL/\mu$ is the Reynolds number, which is based on the characteristic values of the velocity (U), the spatial scale (L), and the viscosity parameter (μ) of the fluid.

Shallow Water Equations:

$$\frac{\partial \mathbf{u}}{\partial \mathrm{t}} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \omega \times \mathbf{u} = -\nabla \mathrm{h} + \frac{1}{\mathrm{R_e}} \nabla^2 \mathbf{u} \tag{2.3}$$

$$\frac{\partial h}{\partial \mathrm{t}} + (\mathbf{u} \cdot \nabla)(1 + \mathrm{h}) + (1 + \mathrm{h}) \nabla \cdot \mathbf{u} = 0 \tag{2.4}$$

where $\mathbf{u} = (\mathrm{u}, \mathrm{v}, 0)$, $\omega = (0, 0, \mathrm{y})$, and h is the sea-surface elevation. The equatorial nondimensionalization scheme [3] was used in deriving the above shallow water equations.

3 Spectral Element Discretizations

The basis sets used in the present work are as follows:

$$\psi^e_{lm}(\xi, \eta) = h_l(\xi) h_m(\eta) \qquad l, m \in \{0, 1, \ldots N\}^2 \quad (2D) \tag{3.1}$$

$$\psi^e_{lmn}(\xi, \eta, \zeta) = h_l(\xi) h_m(\eta) h_n(\zeta) \qquad l, m, n \in \{0, 1, \ldots N\}^3 \quad (3D) \tag{3.2}$$

where $h_i(s)$ are the Gauss-Lobatto-Legendre polynomials.

If we use a single subscript, q ($q \in \{1, 2, \ldots, (N+1)^d\}$), the mapping between a macroelement, Ω_e, and its phase domain, $\hat{\Omega}_e$, can be expressed as:

$$\mathbf{x} = \mathbf{x}_\mathrm{q} \psi^e_\mathrm{q}(\xi) \tag{3.3}$$

Where $\mathbf{x} \in \Omega^e$ and $\xi \in \hat{\Omega}_e$.

Let solution $\mathbf{u}$ at time $n\Delta t$ on each subdomain Ω^e be expanded as:

$$\mathbf{u}^e(\mathbf{x}, \mathrm{n}\Delta\mathrm{t}) = \mathbf{u}_\mathrm{q}^e(\mathrm{n}\Delta\mathrm{t})\psi_\mathrm{q}^e[\xi(\mathbf{x})] \tag{3.4}$$

where $f_q^e(t)$ is the value of function f at the collocation point $\mathbf{x}_\mathrm{q} \in \Omega^e$ at time t.

We use a semi-implicit temporal discretization scheme for the Navier-Stokes equations, i.e., the advection terms are treated with the third-order Adams-Bathforth scheme, and the mixing terms with the Crank-Nicolson scheme. Then, the resulting equations are:

$$\frac{\mathbf{u}^* - \mathbf{u}^\mathrm{n}}{\Delta\mathrm{t}} = \sum_{\mathrm{i}=0}^{2} \alpha_i(\mathbf{u}^{\mathrm{n-i}} \cdot \nabla)\mathbf{u}^{\mathrm{n-i}} \tag{3.5}$$

$$\frac{\mathbf{u}^{\mathrm{n}+1} - \mathbf{u}^*}{\Delta\mathrm{t}} = -\nabla p^{n+\frac{1}{2}} + \frac{1}{R_e}\nabla^2\mathbf{u}^{\mathrm{n}+1} \tag{3.6}$$

$$\nabla^2 p^{n+\frac{1}{2}} = \frac{1}{\Delta t}\nabla \cdot \mathbf{u}^* \tag{3.7}$$

where $\alpha_0 = -\frac{23}{12}$, $\alpha_1 = \frac{4}{3}$, and $\alpha_2 = -\frac{5}{12}$.

Applying the above spectral element discretization scheme to the weak forms of the Navier-Stokes equations, we derive the following matrix formulae:

$$[B][\mathbf{u}^*] = [\mathrm{B}][\mathbf{u}^\mathrm{n}] + \Delta\mathrm{t}\sum_{\mathrm{i}=0}^{2}\alpha_\mathrm{i}\sum_{\mathrm{j}=1}^{\mathrm{d}}[\mathbf{C}_\mathrm{j}^{\mathrm{n-i}}][\mathrm{u}_\mathrm{j}^{\mathrm{n-i}}] \tag{3.8}$$

$$[B][\mathbf{u}^{\mathrm{n}+1}] = [\mathrm{B}][\mathbf{u}^*] - \Delta\mathrm{t}\left\{\mathbf{D}^\mathrm{T}[\mathrm{p}^{\mathrm{n}+\frac{1}{2}}] + \frac{1}{\mathrm{R_e}}[\mathrm{A}][\mathbf{u}^{\mathrm{n}+1}]\right\} \tag{3.9}$$

$$[A][p^{n+\frac{1}{2}}] = -\frac{1}{\Delta t}[\mathbf{D}] \cdot [\mathbf{u}^*] \tag{3.10}$$

where $[A]$ and $[B]$ are the stiffness matrix and mass matrix, respectively; $[\mathbf{C}_\mathrm{j}^\mathrm{n}]$ and $[\mathbf{D}]$ are generalized vectors whose m^{th} components, $[C_{j,m}^n]$ and $[D_m]$, are the advection matrix and gradient matrix, respectively.

The boundary conditions used in the above derivations are:

$$\mathbf{u}(\mathbf{x}, \mathrm{t}) = \mathbf{f}(\mathbf{x}, \mathrm{t}) \qquad \mathbf{x} \in \partial\Omega \tag{3.11}$$

$$p(\mathbf{x}, \mathrm{t}) = \mathrm{g_1}(\mathbf{x}, \mathrm{t}) \quad \text{or} \quad \nabla \mathrm{p}(\mathbf{x}, \mathrm{t}) \cdot \mathbf{n} = \mathbf{g_2}(\mathbf{x}, \mathrm{t}) \qquad \mathbf{x} \in \partial\Omega \tag{3.12}$$

where $\partial\Omega$ is the boundary of the computational domain, Ω.

We chose the isoparametric spectral element discretization scheme, namely, using nonstaggered grids, for the present numerical model. The nonstaggered formulation avoids spurious pressure modes as staggered schemes do, and, at the same time, has the advantage that pressure is continuous across boundaries of the spectral elements. Only one set grids is required for both interpolation and quadrature, hence simplifying operations. We find that the nonstaggered grid formulations do not necessarily entail high costs for the iterative pressure solver as one might expect, and the two types of pressure boundary conditions in (3.12) make little difference in terms of computational efficiency.

The discretized shallow water equations have the following form:

$$[B][\mathbf{u}^{\mathrm{n}+1}] = [\mathrm{B}][\mathbf{u}^\mathrm{n}]$$

$$+\Delta t \sum_{i=0}^{2} \alpha_i \left\{ \sum_{j=1}^{2} [\mathbf{C}_j^{n-i}][\mathrm{u}_j^{n-i}] - [\mathbf{D}^{\mathrm{T}}][\mathrm{p}^{n-i}] - \frac{1}{\mathrm{R_e}}[\mathrm{A}][\mathbf{u}^{n-i}] \right\} \tag{3.13}$$

$$[B][h^{n+1}] = [B][h^n]$$

$$+\Delta t \sum_{i=0}^{2} \alpha_q \left\{ [D_1][u^{n-i}(1+h^{n-i})] + [D_2][v^{n-i}(1+h^{n-i})] \right\} \tag{3.14}$$

Details of deriving Eqs.(3.13) and (3.14) were given in [3].

4 Parallel Algorithm and Implementation

For a distributed-memory, massively parallel computer, which is the chosen architecture for the present models, remote memory access usually is much slower than local memory access. To efficiently implement the spectral element model on this type of computer, we must create a data parallelism which minimizes communication among processing nodes. This goal is achieved through a data-mapping scheme that allows for all the information related to a given spectral element to be collected in the memory of a single processor. On the Connection Machine model CM-5, we pursue data parallelism by designing the layout of the arrays of the spectral element model in such a way that the axes along the number of elements are assigned as parallel dimensions, and those along intra-element degrees of freedom as serial ones. The following is a three-dimensional example in CM Fortran syntax

$$REAL \quad A(N,N,N,K_1,K_2,K_3)$$
$$CMF\$LAYOUT \quad A(:SERIAL,:SERIAL,:SERIAL,:NEWS,:NEWS,:NEWS)$$

where K_1, K_2, K_3 are numbers of elements in each spatial dimension, separately, and N is the order of polynomials.

Basically, there are four types of algebraic computations involved in solving the discretized Navier-Stokes Equations (3.8)-(3.10), when the preconditioned conjugated gradient iteration method is applied:

$$[r] = [A][u] \tag{4.1}$$

$$[s] = [B][u] \quad or \quad [B]^{-1}[u] \tag{4.2}$$

$$[t] = [C_{m,j}^n][u] \tag{4.3}$$

$$\alpha = [u] \cdot [v] \tag{4.4}$$

Operations in (4.1)–(4.3) are all matrix-vector products, except that the matrices in (4.2) and (4.3) are diagonal ones. The elements of $[A]$ and $[B]$ are constants which only need to be calculated once; those of $[C_{m,j}^n]$ are not, and they need to be evaluated at each time step. (4.4) is the inner product of two vectors.

Matrices in (4.1)–(4.3) are global ones, which means that they are the results of the direct stiffness summation. The way in which the direct stiffness summation is performed can significantly affect the efficiency of the parallel implementation of the model. In serial spectral element algorithms, direct stiffness summation usually is carried out automatically by using local and global node numbering systems. However, in data

parallel programs we have to treat direct stiffness summation separately to avoid explicit short messages.

We split the procedure of calculating matrix-vector products into two steps, each of which admits concurrency. At the first step, the matrix-vector products are carried out at the elemental level with, for example, the elemental Laplacian and mass matrices:

$$r^k(i) = \sum_{q=1}^{(N+1)^d} A^k(i,q)u^k(q) \qquad i \in \{1,2,...,(N+1)^d\}, \quad k \in \{1,2,...,K\} \tag{4.5}$$

$$s^k(i) = B^k(i)u^k(i) \qquad i \in \{1,2,...,(N+1)^d\}, \quad k \in \{1,2,...,K\} \tag{4.6}$$

The array layout described at the beginning of this section defines a one-to-one correspondence between the spectral elements and the virtual processors. All data related to a given element are stored in the memory of a single processor. Therefore, there is no communication among neighboring processors during these elemental level computations, and they are performed concurrently across all virtual processors.

After applying tensor-product factorization, the computational complexities to evaluate (4.5) and (4.6) would be C_1KN^{d+1}/Q and C_2KN^d/Q, respectively, where Q is the number of physical processors involved. On the Connection Machine systems, parallel data structure allows (4.6) to be performed in an array operation, which means that thousands of simultaneous multiplications are made across all the array elements. Hence, C_2 is a small number. Consequently, diagonal preconditioning (4.2) is especially efficient in the data parallel environment: it does not require direct stiffness summation, and only local computation is involved, which is very fast. Iteration counts can be reduced by twenty to thirty percent with about a one percent increase in cost.

The processing nodes on the latest CM-5 model are equipped with powerful vector-processing units that can further reduce the cost of elemental level computation. These vector-processing units are most efficient when the order of the spectral elements is high.

The second step is to carry out direct stiffness summation, $\sum_{k=1}^{K}{}'$, in which contributions from local nodes that share the same physical coordinates are first accumulated, and then assigned back to those local nodes. In a serial spectral element model, this procedure can be accomplished by using global and local index systems, and is automatically done as the matrix computation is made for each spectral element. In the parallel spectral element model, however, it is more efficient to use a separate step for the direct stiffness summation. Since each spectral element has at least one edge (two-dimensional case) or one surface (three-dimensional case) that is shared by a neighboring element, the direct stiffness summation can be carried out simultaneously along these edges or surfaces enabling structured message exchange, i.e., edge-based message exchange for two-dimensional problems, and surface-based message exchange for three-dimensional ones. Since this kind of information exchange takes place along the linkages of the "macro-element-skeleton", it can be easily synchronized for all elements in the entire domain. The work per processor that is required in this procedure is C_3dKN^{d-1}/Q. The structured message exchange mostly avoids explicit short messages, and it considerably improves the parallel efficiency of the spectral element model [1].

With parallel prefix of the CM Fortran, MATMUL and SUM, the inner product (4.4) is executed completely in parallel. Its computational complexity is C_4KN^d/Q. Due to the high level of concurrency afforded by the parallel prefixes, C_4 is a small number.

Eq.(3.8) only requires a direct method to solve. The computation kernel here is the evaluation of the advection term where concurrency can be achieved at different levels of the computation. We first evaluate the shears of velocities at all nodal points

$$\frac{\partial u_m^{n,e}(l)}{\partial x_j} = \sum_{s=1}^{(N+1)^d} u_m^{n,e}(s) \frac{\partial \psi_s^e}{\partial x_j}$$
$$l \in \{1, 2, ..., (N+1)^d\}, \quad m, j \in \{1, 2, ..., d\}^2, \quad e \in \{1, 2, ..., K\} \quad (4.7)$$

This operation is executed concurrently across all virtual processors. With the partial summation method, the computational complexity for (4.7) is $C_5 K(N+1)^{d+1}/Q$. The advection term in (3.8) also can be written as

$$C_{m,j}^{n,e}(p,q) = \sum_{s=1}^{(N+1)^d} u_m^{n,e}(s) \frac{\partial \psi_s^e(q)}{\partial x_j} \int_{\hat{\Omega}^e} \psi_p^e \psi_q^e |J^e| d\hat{\Omega}^e$$
$$p, q \in \{1, 2, ..., (N+1)^d\}^2, \quad j, m \in \{1, ..., d\}^2 \quad (4.8)$$

Therefore, once the shears of velocities are obtained, the remaining operation to evaluate the advection terms is the same as that of (4.6). Hence, the total computational complexity of (4.8) is $C_2 K(N+1)^d/Q + C_5 K(N+1)^{d+1}/Q$.

We can undertake similar parallelization analysis for solving the discretized shallow water equations (3.13) and (3.14). In this case, the computational complexity for matrix-vector product and that for the convection term are the same as those derived for the Navier-Stokes equations, except $d = 2$. Since no iterative procedures are involved, the model for the shallow water equations is relatively inexpensive compared to a "true" three-dimensional model.

As spectral elements are of high-order, most of the costly operations are at the elemental level, and they are executed concurrently. The spectral granularity at the elemental level can take full advantage of the computing power that the latest processing units provide. The structured message exchange, combined with parallel prefix, makes inter-element communication a lower-order rather than a highe-order cost, compared to that of elemental level computation. This communication cost should be much smaller than that of the h-type finite element model, partially because many fewer redundant nodel values, shared by more than one element, have to be stored.

Tables 1 and 2 show the CM-5 timing results per pressure iteration in solving the two-dimensional and three-dimensional Navier-Stokes equations, respectively. We find that for a fixed order of the basis functions (N), expanding the size of the problem by increasing K (the number of the spectral elements) and, at the same time, increasing the number of processing nodes by the same proportion, hardly changes the computational cost in terms of CPU time per pressure iteration. Due to the high parallel efficiency, the spectral element models can fully take advantage of the highly scalable performance of the massively parallel architectures.

Table 1: CM-5 Timing Results per Pressure Iteration (2-D)

N	K	Num. Grid Pts.	Num. Nodes*	CPU (s)
7	1024	50625	32	0.062
7	4096	201601	128	0.063
15	1024	231361	32	0.317
15	4096	923521	128	0.318

* Each CM-5 node has 4 vector-processing units and a peak performance rating of 128 MFLOPS.

Table 2: CM-5 Timing Results per Pressure Iteration (3-D)

N	K	Num. Grid Pts.	Num. Nodes	CPU (s)
4	4096	274625	64	0.106
4	32768	2146689	512	0.108
7	4096	1442897	64	0.473
7	32768	11390625	512	0.475

Table 3 shows the parallel performance for the spectral element model for the shallow water equations. To compare the performance of the spectral element model on the CM-5 with that on a sequential computer, say, RISC/6000, we made Table 4 for "equivalent performance", η_{EP}, which is defined as $\eta_{EP} = 28 MFLOPS \times K \times \frac{T_{RISC}}{T_{CM5}}$. K is the number of spectral elements, T_{RISC} is the double precision timing on the RISC/6000 (model 560) which has a LINPACK performance rating of 28 MFLOPS (T_{RISC}=0.012 s per spectral element per time step), and T_{CM5} is the double-precision timing on the CM-5.

Table 3: CM-5 Timing Results (in seconds) per Time Step for the Spectral Element Shallow Water Equation Model (N=16)

K	128 nodes	256 nodes	512 nodes
100	0.31	0.31	0.31
256	0.32	0.31	0.31
4900	0.66	0.48	0.32
10000	0.83	0.48	0.32
22500	1.71	0.97	0.66
40000	2.33	1.29	0.80
90000	-	-	1.6

Table 4: Equivalent Performance for CM-5 (in MFLOPS) for the Spectral Element Shallow-Water-Equation Model (N=16)

K	128 nodes	256 nodes	512 nodes
100	115	115	115
256	2789	2789	2879
4900	2650	3644	5466
10000	4301	7437	11156
22500	4725	8280	12170
40000	6208	10984	17850
90000	-	-	20081

One advantage of the high-order domain decomposition numerical model is that it can take better advantage of the vector-processing units than can low-order models. Table

5 shows that the difference between the performance of the spectral element model with vectorization and that without vectorization increases as the order of the basis functions, N, increases.

Table 5: CM-5 Timing Results per Time Step for the 2-D N-S Model (32 nodes) With an Iteration Error of $O(10^{-6})$

		CPU (s)	
N	K	With Vectorization	Without Vectorization
3	576	2.2	3.2
4	100	2.1	4.8
5	16	1.9	7.9
6	4	2.6	11.4

5 CONCLUSIONS

In present work, we have shown that high-order domain decomposition methods can be efficiently applied in a data parallel programming environment. The optimized computational efficiency of the parallel spectral element model comes not only from the exponential convergence of its numerical solutions, but also from its efficient usage of the powerful vector-processing units of the latest parallel architectures which have a highly scalable performance. The nonstaggered grid formulation is convenient for, and shows no disadvantage in, these parallel spectral element models.

REFERENCES

1. Fischer, P. F. and A. T. Patera, J. Comput. Phys., 92 (1991) 380.
2. Korczak, K. Z. and A. T. Patera, J. Comput. Phys., 62 (1986) 361.
3. Ma, H., J. Comput. Phys., 109 (1993) 133.
4. Maday, Y. and A. T. Patera, in *State-Of-The-Art Surveys on Computational Mechanics,* A.D. Noor and J. T. Oden (eds.), New York, New York, 1989.
5. Patera, A. T., J. Comput. Phys., 54 (1984) 468.
6. Ronquist, E. M., Optimal Spectral Element Methods for the Unsteady Three Dimensional Navier-Stokes Equations, Ph.D. Thesis, The Massachusetts Institute of Technology (1988).

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A Parallel Implementation of a Spectral Multi-domain Solver for Incompressibile Navier-Stokes Equations

G. De Pietro [a] A. Pinelli [b] and A. Vacca [c]

[a]IRSIP, CNR, Via P.Castellino, 111, 80128 Naples, Italy

[b]School of Aeronautics, Polytechnic University of Madrid, Spain.

[c] II University of Naples, Aversa, Italy.

1. Introduction

In the last years domain decomposition methods have gained much attention in the CFD comunity. One of the most relevant features of such methods is concerned with the possibility of tuning the accuracy of the numerical discretization according to the expected behaviour of the solution in each subdomain. Consequently, subregions of flow field containing sharp boundary layer, can be enclosed within subdomains with high resolution, while low resolution can be assigned to subregions where smooth solutions can be expected.

These advantages can be fully exploited when discretizing the equations with spectral methods which guarantee a fast decay of the error with the number of the nodes, termed as "spectral accuracy".

On the other hand domain decomposition methods might provide a natural stabilization strategy for the spectral discretization which is a "central one" in nature. In fact the local cell Peclet number can be locally diminished by reducing the mesh spacing within the critical subdomain, without the introduction of any particular stabilization procedure.

From the computational point of view, the domain decomposition techniques is well suited for parallel computing, even if in practical case several difficulties arises whenever good performances have to be reached [1].

In this paper, a parallel algorithm for the solution of the bidimensional incompressible Navier-Stokes equations is presented. After a brief introduction of the time splitting scheme used for the time discretization of the unsteady incompressible Navier-Stokes equations, the attention will be mainly focused on the the spectral multidomain approach and on its parallel features. Finally, performance results concerning the parallel implementation on two different MIMD parallel architectures will be presented.

2. Navier-Stokes Equations and Time Splitting Scheme

When the incompressible Navier-Stokes equations

$$\frac{\partial U}{\partial t} + \frac{1}{2}\left(U \cdot \nabla U + \nabla \cdot (UU)\right) = -\nabla p + \frac{1}{Re} \triangle U \qquad (1)$$

$$\nabla \cdot U = 0 \tag{2}$$

are solved by means of a projection method [2], with the diffusive terms treated in an implicit fashion [3], the time stepping procedure consists in a cascade of scalar elliptic kernels, to be solved at each time step. Namely two (for the two-dimensional equations) Helmohltz problems for the inversion of the diffusive part, and a Poisson problem for the pressure need to be solved at each time step. It is then clear that, in order to achieve a globally efficient algorithm, it is of fundamental importance to tackle effectively the mentioned scalar problems.

For the sake of completeness in the following the adopted fractional step scheme (i.e. Van Kan's pressure correction method [4])is given

$$\frac{\hat{U} - U^n}{\Delta t} - \frac{1}{2Re} \triangle \left(\hat{U} + U^n\right) = -\nabla p^n - \frac{3}{2}\mathcal{L}\left(U^n\right) + \frac{1}{2}\mathcal{L}\left(U^{n-1}\right) \tag{3}$$

$$\hat{U}|_{\partial\Omega} = U\left((n+1)\,\Delta t\right) \tag{4}$$

$$\frac{U^{n+1} - \hat{U}}{\Delta t} + \frac{1}{2}\nabla\left(p^{n+1} - p^n\right) = 0 \tag{5}$$

$$\nabla \cdot U^{n+1} = 0 \tag{6}$$

where $\mathcal{L}(U)$ represents the advective term $\frac{1}{2}(U \cdot \nabla U + \nabla \cdot (UU))$.

In the first step, a non physical intermediate velocity field $\hat{U}$ is computed. In fact, $\hat{U}$ does not satisfy the incompressility condition. Then in the second step $\hat{U}$ is projected onto the divergence free space to get an adeguate velocity approssimation of U^{n+1}.

The scheme with the given boundary conditions is nothing else then a second order Crank-Nicolson Adams-Bashforth scheme with an $\mathcal{O}(\Delta t^2)$ deviation in the tangent direction of the boundary. By applying the divergence operator to (6), it turns out that the latter is equivalent to

$$\triangle\left(p^{n+1} + p^n\right) = \frac{2}{\Delta t}\nabla \cdot \hat{U} \tag{7}$$

$$\frac{\partial p^{n+1}}{\partial \mathbf{n}}|_{\partial\Omega} = 0 \tag{8}$$

$$U^{n+1} = \hat{U} - \frac{\Delta t}{2}\nabla\left(p^{n+1} - p^n\right) \tag{9}$$

In the next section the attention will be focused on the way each scalar elliptic problem has been tackled in the framework of a spectral multidomain discretization.

3. Space Discretization and Projection Decomposition Method

In the present work, a Legendre spectral collocation technique coupled with a domain decomposition method has been used for the space discretization of the differential equations. Additional references can be found in [6], [5] for the projection decomposition method, and in [7] for the spectral approximation method.

The following problem, rappresentative of one of the elliptic scalar problems mentioned in the previous section, is considered hereafter:

$$-\triangle u + \alpha u = f \quad \text{in } \Omega, \quad f \in L^2(\Omega) \tag{10}$$

$$u = 0 \quad \text{on } \partial\Omega \tag{11}$$

where α is a real constant ≥ 0, and where Ω is an open connected set $\Omega \subset \Re^2$; in particular, $\overline{\Omega} = \cup_{i=1}^{N} \overline{\Omega_i}$ with $\overline{\Omega_i}$ is a closed rectangle having either common side or common vertex with each neighbour; $\alpha \geq 0$ is either identically equal to zero (i.e., for the Poisson problem related with the pressure) or is equal to $2/\Delta t Re$ (i.e., for one of the momentum equations), and the equivalent weak formulation of (10), (11) is:

$$\begin{aligned} &\text{find } u \in H_0^1(\Omega) \qquad \text{such that} \\ &\qquad\qquad l(u,v) = (f,v)_{L^2(\Omega)} \quad \forall\, v \in H_0^1(\Omega) \end{aligned} \tag{12}$$

where $H_0^1(\Omega)$ is the real Hilbert space defined as follows:

$$H_0^1(\Omega) \equiv \{u \in L^2(\Omega) : \frac{\partial u}{\partial x_1} \in L^2(\Omega) \text{ and } \frac{\partial u}{\partial x_2} \in L^2(\Omega),\ u|_{\partial\Omega} = 0\} \tag{13}$$

equipped with the scalar product:

$$l(u,v) = \int_\Omega (\nabla u \cdot \nabla v + \alpha uv) d\Omega \qquad \forall u, v \in H_0^1(\Omega) \tag{14}$$

Following the classical domain decomposition technique problem (12) is decoupled into a set of problems within each subdomain plus an additional problem at the interfaces Γ:

$$\Gamma = (\Omega \backslash \Omega_0) \backslash \partial\Omega \quad \text{with} \quad \Omega_0 = \cup_{i=1}^{N} \Omega_i \tag{15}$$

Let $H_0^{1/2}(\Gamma)$ be the completion of the normed vector space S defined as:

$$\begin{aligned} S &\equiv \{z \in C^0(\Gamma) : \exists \phi \in C_0^\infty(\Omega) \text{ such that } z = \phi_\Gamma\} \\ \|z\| &= \inf_{\substack{\phi \in C_0^\infty(\Omega) \\ \phi_\Gamma = z}} \|\phi\|_{H_0^1(\Omega)} \end{aligned} \tag{16}$$

where ϕ_Γ is the restriction of ϕ on Γ.

The linearity and continuity of the operator

$$\phi \in C_0^\infty(\Omega) \rightarrow \phi_\Gamma \tag{17}$$

into $H_0^{1/2}(\Gamma)$ and the fact that $C_0^\infty(\Omega)$ is dense in $H_0^1(\Omega)$ leads to the existence and uniqueness of a linear and continuous operator γ (trace operator) from $H_0^1(\Omega)$ onto $H_0^{1/2}(\Gamma)$ defined as

$$\gamma\phi = \phi_\Gamma \quad \forall \phi \in H_0^1(\Omega) \tag{18}$$

The γ operator allows to identify two closed mutually orthogonal subspaces

$$K \equiv \ker(\gamma) = \{u_0 \in H_0^1(\Omega) : \gamma u_0 = 0\} \tag{19}$$

where $\ker(\gamma)$ is the kernel of operator γ, and its orthogonal complement $K^\perp$ is defined as:

$$K^\perp \equiv \{\tilde{u} \in H_0^1(\Omega) : l(\tilde{u}, v_0) = 0 \quad \forall\, v_0 \in K\} \tag{20}$$

Therefore, the solution $u \in H_0^1(\Omega)$ of problem (12) can be uniquely decomposed as

$$u = u_0 + \tilde{u}, \quad u_0 \in K \quad \text{and} \quad \tilde{u} \in K^\perp \tag{21}$$

Since the restriction γ_0 of the operator γ to $K^\perp$ is an isometric isomorphism between $K^\perp$ and $H_0^{1/2}(\Gamma)$ it follows that

$$\forall\ \tilde{u} \in K^\perp \quad \exists!\ \psi \in H_0^{1/2}(\Gamma):\ \ \tilde{u} = \gamma_0^{-1}\psi \tag{22}$$

Identity (21) can be reformulated as:

$$u = u_0 + \gamma_0^{-1}\psi \quad \text{with} \quad u_0 \in K \quad \text{and} \quad \psi \in H_0^{1/2}(\Gamma) \tag{23}$$

Thus, problem (12) can be easily proven to be equivalent to the set of the two following ones:

Problem (P1): find $u_0 \in K$ such that

$$l(u_0, v_0) = (f, v_0)_{L^2(\Omega)} \qquad \forall\, v_0 \in K \tag{24}$$

Problem (P2): find $\psi \in H_0^{1/2}(\Gamma)$ such that

$$l(\gamma_0^{-1}\psi, \gamma_0^{-1}z) = (f, \gamma_0^{-1}z)_{L^2(\Omega)} \quad \forall\, z \in H_0^{1/2}(\Gamma) \tag{25}$$

Problem P1 is nothing else than the solution of N decoupled elliptic problems with homogeneous Dirichlet boundary conditions on both $\partial\Omega$ and Γ.

As concern problem P2, if $\{\xi_i\}\ \ i = 1, .., \infty$ is a set of linearly independent functions which constitute a base for $H_0^{1/2}(\Gamma)$, then the discrete version of problem P2 reads as:

$$l(\gamma_0^{-1}\sum_{i=1}^{M} a_i\xi_i, \gamma_0^{-1}\xi_j) = (f, \gamma_0^{-1}\xi_j)_{L^2(\Omega)} \quad \forall\, j = 1, .., M \tag{26}$$

Typically M corresponds exactly to the number of points on the interface. To set up an algebraic equivalent of (26) the operator γ_0^{-1} should be explicitly formulated. In practice, the operator γ_0^{-1} is never required if an iterative procedure is introduced to solve problem P2. To illustrate this point, it must be remarked that $\tilde{u}^k \in K^\perp$ must satisfy the orthogonality condition:

$$l(\tilde{u}^k, v_0) = 0 \qquad \forall\, v_0 \in K \tag{27}$$

which corresponds to the solution of N elliptic problems (24) with Dirichlet boundary conditions: homogeneous on $\partial\Omega$ and to be iteratively determined on Γ.

To provide at each iteration k the condition on Γ for problem (27) the Green's formula is applied to (26)

$$R^k = \int_\Gamma \frac{\partial \tilde{u}^k}{\partial n}\xi_j d\Gamma - \int_\Gamma \frac{\partial u_0}{\partial n}\xi_j d\Gamma \qquad \forall\, j = 1, ..M \tag{28}$$

where $\tilde{u}^k = \gamma_0^{-1}\sum_{i=1}^{M} a_i^k\xi_i$ is the solution at iteration k of problem (27), where $\frac{\partial}{\partial n}$ represents the jump of the normal derivatives on Γ. R^k is the residual at iteration k, from which the updating of the boundary value $\tilde{u}^{k+1}|_\Gamma$ can be obtained within the chosen iterative procedure.

The convergence rate of the iterative procedure strongly depends on the choice of the basis $\{\xi_i\}$ [8]. For the present work the basis functions proposed by Ovtchinnikov [8] have been used. These constitute a nearly optimal basis, in the sense that the condition number of system (26) is bounded by a constant independent of M, where M is the dimension of the subspace of $H_0^{1/2}(\Gamma)$ generated by $span\{\xi_i\}\ \ i = 1, .., M$.

In view of the character of the algebraic problem (symmetric positive defined) the conjugate gradient has been employed to solve problem (26).

4. Extension to Navier-Stokes equations

If each single differential problem is tackled with the algorithm described in the previous section, at the end of each time step the solution is equivalent to one, hypotetically achieved by solving the whole domain at once.

The last statement requires some comments. When finite dimensional approximation of the space where the solution is sought are considered numerical problem might arise within the fractional step algorithm in the interface neighbouring regions. In particular when the projection step (5) is considered, a straigh use of the results obtained with the present multidomain method leads a discontinuous value of the divergence free velocity field along the interfaces.

From the numerical point of view, these discontinuities, even if limited to a set of measure zero (Γ) might introduce an artificial "numerical boundary layer" that the whole time integration procedure cannot damp out and that might lead to catastrophic instabilities. To avoid such a drawback two solutions are possible. The first one relies upon increasing the dimension of the approximation subspaces to reduce the jumps at the interface. The second one consists in replacing the gradient of the pressure in equation (5) with an equivalent function in $L^2(\Omega)$, which differs from the original one only along sets of measure zero. In particular, the gradient $\mathcal{Q} = \nabla(\phi^{n+1} - \phi^n)$ of the solution, achieved by solving (7) with the previously outlined multi-domain spectral method is substituted with the vector function $\mathcal{G}$ defined as:

$$\mathcal{G}_i(x,y) = \begin{cases} \mathcal{Q}_i(x,y) & \forall\,(x,y) \in \overline{\Omega}\backslash\Gamma \\ \frac{1}{\omega_j^r+\omega_j^s}[\mathcal{Q}_i^r\omega_j^r + \mathcal{Q}_i^s\omega_j^s] & \forall\,(x,y) \in \Gamma_{rs} \end{cases} \quad \forall\, i,j = 1,2 \tag{29}$$

where $\Gamma_{rs} = \overline{\Omega_r} \cap \overline{\Omega_l}$, ω_j^r (ω_j^s) is the Gauss-Legendre quadrature weight along Γ_{rs} (either j or i) corresponding to the node (x,y) in the subdomain $\overline{\Omega}_r$ ($\overline{\Omega}_s$) and $\mathcal{Q}^r$ ($\mathcal{Q}^s$) is the restriction of $\mathcal{Q}$ in $\overline{\Omega}_r$ ($\overline{\Omega}_s$) evaluated in (x,y).

5. Parallel implementation

As concerns the parallel implementation of the given algorithm, we have used a slightly modified version of master-slave computational model. In particular, the major difference with respect to the classical model is that our master actively cooperates with the slaves during the calculation phase, while in the standard version, the master is only demanded to distribute initial data and to gather the results. In our implementation, the activities are shared between master and slaves as follow. At the beginning of the computation, the master process calculates the guess values for the Dirichlet problems. These values are then trasmitted to the slave processes: each of the slaves solves the Dirichlet problems for the assigned domains; it should be noted that, in this case, the domain decomposition (which allow the slaves to operate in parallel) derives directly from the multi-domain approach. After this first phase, the slaves transmit the calculated values at the domain interfaces to the master, which calculates the new values by applying a Conjugate Gradient algorithm, and communicates these values to the slaves for the next iteration. The main causes of inefficiency in using parallel architectures are an uneven load-balancing and the communication overheads. In general, the multidomain technique can generate load

balancing problems because the size and/or computation of blocks can widely differ; however, in our case each domain has the same number of points. Thus, if the number of domains is a multiple of the number of processor, we obtain an optimal load balancing. The communication overheads is mainly related to the Conjugate Gradient algorithm: at each time iteration, data need to be exchanged between processors containing adjacent domain interfaces and the master processor. Because of the sequantial flow of these activities, it is not possible to overlap computation and communication, so the time spent for these communications can represent a not negligible part of the overall computing time.

The parallel version of the code has been developed for message passing environments. In particular, the code has been written in Fortran 77 plus PVM 3.3 communication primitives. In order to meet the goal of overlapping computation and communication, non-blocking communication primitives have been used. Note that the parallelism is exploited only among slaves: the master and the slaves cannot operate in parallel. Anyway, as the great part of the computation is demanded to the slaves, the obtained performances on various homogeneous parallel systems are quite good.

6. Experimental Results

For the tests, we have used two different parallel machines. The first is a CONVEX C210-MPP0 with a vector processor and 4 scalar processor HP 730 connected via FDDI. The second machine is a MEIKO CS2 with 18 super-Sparc processors connected through a switching network. Both the machine are distributed memory MIMD parallel computers. The tests have been performed by using a number of domain multiple of the number of processors used, so that load balancing is guaranteed. Hence, the cause for the loss of efficiency are the time t_c spent for the communication and the idle time t_w of the slaves waiting for the master results. Note that, while the time t_w is indepenedent of the number N of processors, the time t_c increases according to N: so, for a given problem, a linear decrease of the efficiency is expected.

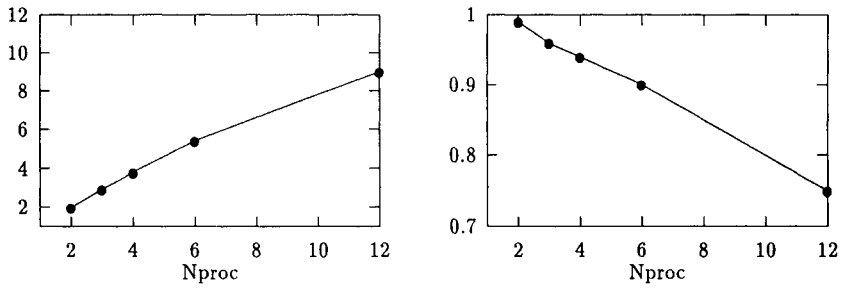


Figure 1. 12 domains with 15×15 points: Speed-up on the left, Efficiency on the right

In figg. (1), (2) the results obtained on the Meiko machine are shown. Note that the values of efficiency are quite good, expecially when the number of points for each domain increases. Moreover, when the number of processor grows the efficiency linearly decrease, as expected.

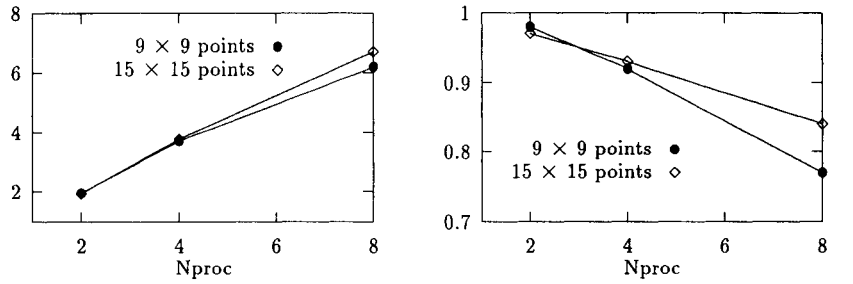


Figure 2. 16 domains: Speed-up on the left, Efficiency on the right

Figure 3 shows a comparison of the results obtained for both the Meiko CS2 and the Convex MPP0 machines. It should be noted that the Convex machine performs better than Meiko when two processors are used; on contrary, by increasining the number of processors the performances of the Meiko are better. This behaviour is essentially related to the different characteristics of the interconnection networks; the FDDI network of the Convex allows very fast communication between two processors at time (the optical fiber is a common shared resource). On the other hand, the CS2 switching network allows to simultaneously execute different communications, so reducing the overall communication time (as matter of fact, also the presence of properly designed communication processors which handle the communication on behalf of the sparc processor has to be taken into account).

To further reduce the computing time, we have also used heterogeneous systems. In fact, whenever the execution of different tasks constituting the same program is strictly sequential, heterogeneous processing can help in enhancing performance by placing a task on the most suitable machine for that task. To this goal, tests have been performed by placing the master process on a vector computer for a more efficient calculation, and the slave processes on a parallel homogeneous system with scalar processors.

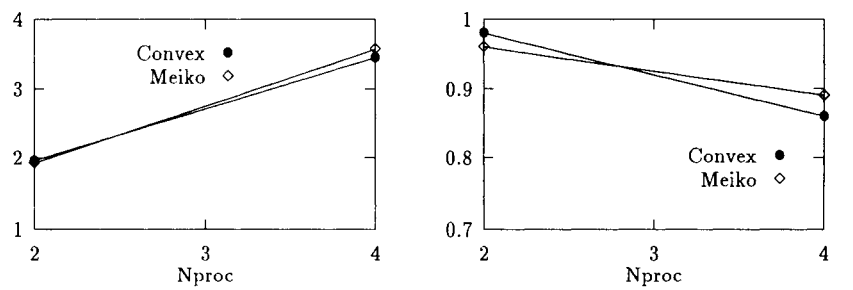


Figure 3. 8 domains with 11×11 points: Speed-up on the left, Efficiency on the right

However, in our case the time spent by the master is a negligible part of the total comuputing time; so, the test performed by using an heterogeneous environment have shown no appreciable improvments.

7. Conclusion

The present work has been concerned with the solution of the unsteady incompressible Navier-Stokes equations, using a high order collocated spectral multi-domain method. The rationale behind the choice and development of the method is given both by the possibility of coupling the potential high accuracy of spectral methods with the flexible framework offered by multi-domains methods, and with the natural way in which a parallel implementation of the present algorithm can be achieved.

In particular, we have shown how the developed algorithm allows for the solution of completely independent and balanced sub-problems leading to full exploitation of MIMD parallel computers.

REFERENCES

1. C. de Nicola, G. De Pietro and M. Giuliani, An Advanced Parallel Multiblock Code for the Solution of 2-D Flow-Field, Lecture Notes in Computer Science, 796, (1994), 139.
2. A.Chorin, A. and Marsden, A Mathematical Introduction to Fluid Mechanics, Springer-Verlag, New York, 1979.
3. A. Pinelli and A. Vacca, Chebyshev Collocation Method and Multidomain Decomposition for the Incompressible Navier-Stokes Equations, Int. J. Num. Meth. in Fluids, 18, (1994), 781.
4. J. Van Kan, A Second Order Accurate Pressure-Correction Scheme for Viscous Incompressible Flow, J. Sci. Stat. Comp., 7, (1986), 870.
5. V. Agoshkov and E. Ovtchinnikov, Projection Decomposition Method, CRS4 Tech. Rep. Cagliari, Italy, 1993.
6. A. Quarteroni, Mathematical Aspects of Domain Decomposition Methods, European Congress of Mathematics, F. Mignot (eds.), Birkhauser Boston, 1994.
7. C. Canuto, M.Y. Hussaini, A. Quarteroni and T.A. Zang, Spectral Methods in Fluid Dynamics, Springer-Verlag, New York, 1988.
8. E. Ovtchinnikov, On the Construction of a Well Conditioned Basis for the Projection Decomposition Method, CRS4 Tech. Rep. Cagliari, Italy, 1993.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

A PARALLEL SPECTRAL ELEMENT METHOD FOR COMPRESSIBLE HYDRODYNAMICS

Anil E. Deane [a*]

[a]Institute for Computational Science and Informatics,
George Mason University &
High Performance Computing Branch,
NASA Goddard Space Flight Center
deane@laplace.gsfc.nasa.gov

We describe results for a Spectral Element-Flux Corrected Transport code that has been parallelized to run on the Convex SPP-1000 Exemplar shared memory parallel computer. The technique considers the domain to be decomposed into multiple regions within each of which the representation is spectral in nature. The regions are patched together by the use of interface conditions. The technique is applied to the compressible Euler equations of hydrodynamics and is well suited for problems when discontinuities such as shocks or contact discontinuities are present in the solution. Since the present code is one-dimensional (a multidimensional version being underway), additional work representing a multidimensional workload is created for the threads by increasing the (polynomial) order of the elements, N. Good scaling for the algorithm is found only for large N on the SPP-1000.

1. Introduction

Spectral element methods [1] combine the flexibility of finite element methods and the spectral accuracy ("infinite-order" accuracy) of spectral methods. The physical domain is divided into several regions in each of which the solution is represented by local basis functions (typically Legendre or Chebyshev interpolating polynomials). These spectral elements are then patched together through techniques common to finite elements to provide a global representation of the flow. The method has found good use in complex geometry fluid flow simulations. Changes of geometry do not lead to fundamental rethinking of appropriate basis, coordinate mapping functions and stability/accuracy considerations. Yet another, considerable, advantage is that since each element represents the solution by local basis functions with patching conditions at their interfaces the method parallelizes in a straightforward manner. Indeed parallel efficiency, scalability, and other measures of relevance to parallel computation are excellent [2].
One deficiency in the method formulated until fairly recently is that of the limitation of the application to incompressible flows. Spectral methods suffer from their poor rep-

*Research supported by NASA HPCCP through NAG 5-2652.

resentation of discontinuities (Gibbs phenomena in particular, and other phase errors for discontinuities); hence spectral element methods, which use spectral representations locally, inherit these disadvantages. Thus compressible flows have not received much attention, although there have been recent investigations. These new studies have focused, appropriately enough, on flux limiting type algorithms and spectral filtering techniques [3–6].

Here we describe a germinal effort in the construction of a parallel spectral element method for compressible flows. It relies heavily on the previously cited work. We discuss the method in some detail, and those issues related to its parallelization. As a testbed machine we choose to implement the algorithm on the Convex SPP-1000 computer, a relatively new machine about which few performance studies are available and which is architecturally interesting. The FCT method using finite differences instead of spectral elements has been previously parallelized on several machines using a number of message passing libraries by [7].

2. Application to hyperbolic conservation laws.

Consider the conservation law,

$$\frac{\partial w}{\partial t} + \frac{\partial f}{\partial x} = 0, \tag{1}$$

where the vector function w represents density, momentum and total energy, while f is the flux function (giving the 1-D Euler equations). i.e.

$$w = (\rho, \rho V_x, \rho e), \qquad f = (\rho V_x, \rho V_x^2 + p, \rho V_x e + p V_x). \tag{2}$$

Integrating the equation once we obtain the canonical semi-discrete flux form:

$$\frac{\partial w}{\partial t} + \frac{1}{\Delta x}(F_{i+1/2} - F_{i-1/2}) = 0, \tag{3}$$

where it is understood that w is now a *cell-averaged* quantity. The quantity F is an *edge based* quantity derived from the cell-averaged f; this involves a spatial averaging procedure and hence we turn to the spatial discretization.

3. Spectral element discretization.

We will be concerned with cell-averaged quantities and *edge* or *point* quantities. The cell-averaged quantities are defined on the grid j for cell-centered quantities,

$$x_j, \quad j = 0, N-1 \quad \text{Gauss} - \text{Chebyshev}, \tag{4}$$

while the edge quantities are defined over the grid i,

$$x_i = \cos\frac{\pi i}{N} \quad i = 0, N \quad \text{Gauss} - \text{Lobatto} - \text{Chebyshev}. \tag{5}$$

The i grid straddles the j grid as shown in Figure 1. Consider the spatial domain parti-

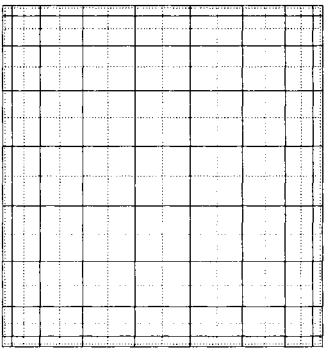


Figure 1. The nodes for the Gauss-Chebyshev points (dashed lines), used for the cell-averaged quantities, and the Gauss-Lobatto-Chebyshev points (solid lines), used for edge quantities, for $N = 9$.

tioned into K elements where in each element, indexed by k,

$$\phi^k(x) = \sum_{i=0}^{N} \phi_i^k h_i(x). \tag{6a}$$

Averaging over space gives,

$$\bar{\phi}^k(x) = \sum_{i=0}^{N} \phi_i^k \bar{h}_i(x), \tag{6b}$$

where a unique (Lagrange) interpolating function h_i can be found:

$$h_i(x) = \frac{2}{N} \sum_{p=0}^{N} \frac{1}{c_p c_i} T_p(x_i) T_p(x), \quad 0 \le i \le N, \tag{7a}$$

and correspondingly,

$$\bar{h}_i(x) = \frac{2}{N} \sum_{p=0}^{N} \frac{1}{c_p c_i} T_p(x_i) \bar{T}_p(x), \quad 0 \le i \le N, \tag{7b}$$

with $c_n = 1$ if $n \neq 0, N$ and $c_n = 2$ otherwise. Here,

$$\bar{T}_0 = 1, \quad \bar{T}_1 = \frac{1}{2}\alpha_1 U_1(x), \quad \bar{T}_i = \frac{1}{2}[\alpha_i U_i(x) - \alpha_{i-2} U_{i-2}(x)], \quad i \ge 2, \tag{8a}$$

and

$$\alpha_i = \frac{\sin[(i+1)\pi/2N]}{(i+1)\sin(\pi/2N)}. \tag{8b}$$

$U_i(x)$ are Chebyshev polynomials of the second kind. The cell-averaging procedure correctly converts point values defined over $N+1$ points to cell-centered average values defined over N points. Note that $h_i(x)$ is an interpolating polynomial, while $\bar{h}_i(x)$ is not. Analogous to (6a), there exists a (Lagrange) interpolant taking cell-averaged quantities into a continuous function,

$$\phi^k(x) = \sum_{i=1}^{N} \bar{\phi}_j^k g_j(x). \tag{9}$$

To recover the edge values from the cell-averaged quantities, the following reconstruction is applied:

$$\phi_i = G_j(x_i)\bar{\phi}_j, \quad \text{where} \quad G_j(x) = \sum_{p=0}^{N-1} \frac{\lambda_p^j}{\alpha_p} U_p(x), \tag{10a}$$

with

$$\lambda_p^j = \frac{1}{N} T_p(x_j), \quad p = N-2, N-1; \quad \lambda_p^j = \frac{1}{N}[T_p(x_j) - T_{p+2}(x_j)], \quad 0 \le p \le N-3 \tag{10b}$$

Note that the reconstruction gives only N points, but the interpolating polynomial constructed for the collocation method requires $N+1$ values. An additional constraint in order to uniquely determine the polynomial is to require C^0 continuity across the interface of the elements so that $\phi_0^{k+1} = \phi_N^k$.
Hence,

$$\phi^{k+1}(\xi) = \sum_{j=1}^{N} \bar{\phi}_j^{k+1} G_j(\xi) + \delta\phi^k, \tag{11a}$$

where

$$\delta\phi^k = (1-\xi) T_N'(\xi) \frac{\phi_N^k - \sum_{j=1}^{N} \bar{\phi}_j^{k+1} G_j(-1)}{2N^2}. \tag{11b}$$

In (11a,11b) we have written (ξ, η, ζ) as the local element coordinate system.
Figure 2a shows the a sinusoid over 32 Gauss points. By use of (6a) its cell-averaged values $\bar{a}(x)$ are also shown. Figure 2b shows the reconstruction of the function via (11a,11b) by plotting the error between the function and its reconstruction. As is evident, the errors are very small.

4. Flux Corrected Transport

The FCT procedure for solving (3) is as follows [8–10]:
1. Form a *low-order flux*: $F_{i+1/2}^L$
2. Form a *high-order flux*: $F_{i+1/2}^H$
3. Form an *anti-diffusive flux*:

$$A_{i+1/2} = F_{i+1/2}^H - F_{i+1/2}^L$$

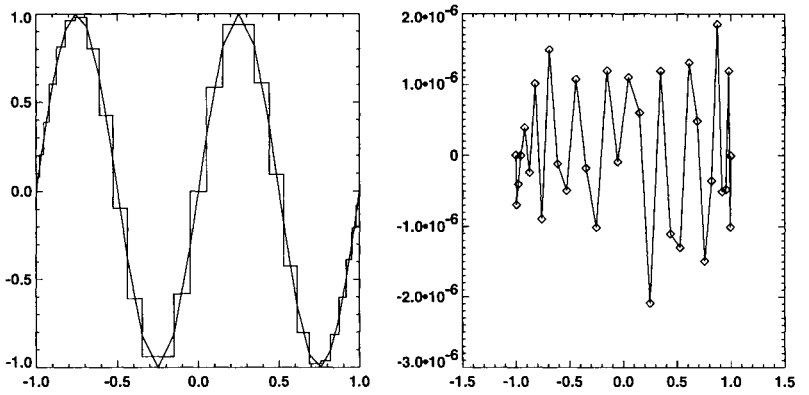


Figure 2. (a) The function $a(x) = \sin 2\pi x$ and its cell average, $\bar{a}(x)$, and (b) The error in the reconstruction from $\bar{a}(x)$ for N=32.

4. Form a *low-order solution*:

$$w_i^L = w_i^n - \frac{\Delta t}{\Delta x}(F_{i+1/2}^L - F_{i-1/2}^L)$$

5. Limit the anti-diffusive flux to prevent spurious extrema:

$$A_{i+1/2}^C = C_{i+1/2} A_{i+1/2}$$

6. Update the solution:

$$w_i^{n+1} = w_i^L - \frac{\Delta t}{\Delta x}(A_{i+1/2}^C - F_{i-1/2}^C)$$

In the Spectral Element-FCT approach we define the cell-centered quantities to be on the Gauss-Chebyshev points (grid j) while the fluxes are on the Gauss-Lobatto-Chebyshev points (grid i). Step 2 above is interpreted in the Spectral Element formulation. Fluxes are obtained via (11a,11b) from the cell-centered flux function as in (2).

5. Shock tube results

We solve a one-dimensional hydrodynamics problem — the shock tube — with parameters due to Sod [11]. At time $t = 0$ the gas on the left of a diaphragm is at pressure $p = 1, \rho = 0.1$ while on the right it is at $p = 1, \rho = 0.125$. Following the rupture of the diaphragm at $t = 0^+$, three waves propagate through the gas: an expansion fan, a contact discontinuity and a shock wave. Figure 3 shows the results obtained with the spectral-element FCT method with $N = 21, K = 20$. While not exceptional, the method does quite well. Some small excrescences are evident.

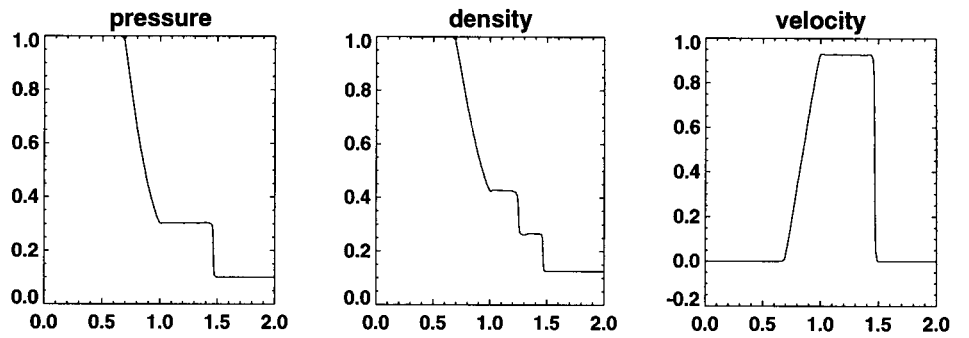


Figure 3. Shock tube results. Time is 0.263, $N = 21$, $K = 20$. Parameter are those due to Sod.

6. Parallelization on the SPP-1000

The Convex SPP-1000 is a fairly new machine[12,13] based on the HP PA-RISC 7100 (100 MHz) microprocessor with 1MB data and instruction caches. The machine possesses a uniform address space. Two processors form a functional unit while 4 functional units are closely coupled into a (hyper-)node. NASA/GSFC's machine has 2 nodes, consisting of a total of 16 processors with 1 GB of memory. The nodes are partitioned into two partitions of 1 and 15 processors respectively. While the machine is similar to the KSR-1 and the T3D, it is unique in that cache coherence is maintained via a directory mechanism within nodes and "Scalable Coherent Interface" hardware across nodes.

The SPP-1000 supports both a shared memory and a message-passing (hosted PVM) model. The message passing primitives lay on top of the shared memory hardware and hence are not as efficient. Therefore we have used the shared memory model invoked with compiler directives. These parallel directives consist of `LOOP_PARALLEL` and `PREFER_PARALLEL`. Loops over spectral elements are distributed typically via the latter directive; however some loops that the compiler did not parallelize (due to indeterminate dependencies) were forced via the former directive. Thus the low-order and high-order flux evaluations, the flux limiting, and solution update are all done in parallel. The boundary conditions and the time step determination are currently done serially. Temporary variable data is declared `LOOP_PRIVATE`. The SPP-1000 also allows the use of `THREAD_PRIVATE` and `NODE_PRIVATE` data types, allowing the data not to be corrupted across thread or node boundaries. It also provides more rapid access of the data. While presently we do not use these data types, later versions of the code will incorporate them, for instance in the time step determination.

7. Performance results

Figure 4 shows the wall clock time taken for the execution of 10 steps of the code versus number of processors for several values of N. Clearly, for small values of N, there is not enough work for the threads and the initiation time of the threads dominates. For large values of N the scaling is better, the curve not turning over until as many as 10 processors, but still does not obtain utilization over the entire number of processors. Interestingly, the 7-8 processor division which crosses the node boundary does not give any special turn-over.

The SPP-1000 has been measured [12] to cost $T_{\rm th} = (9 + 34(n_{\rm th} - 1))$ μsecs, where $n_{\rm th}$ is the number of threads in fork-join operations. For this code there are 20 loops that are to be distributed per half-step; and therefore for 10 complete steps $400T_{\rm th}$ is the approximate cost for thread initiation and termination. For 8 processors this is roughly 0.1 sec. From Figure 4 the $N51$ run is certainly affected by this cost. The turnoff for $N > 51$ in Figure 4 occurs at much higher execution times; hence the turnover is due to non-local memory references and cache misses.

Another option to increase the workload of the 1D code is to increase K, the number of elements. We have chosen to increase the workload by increasing N since the work increases quadratically with N.

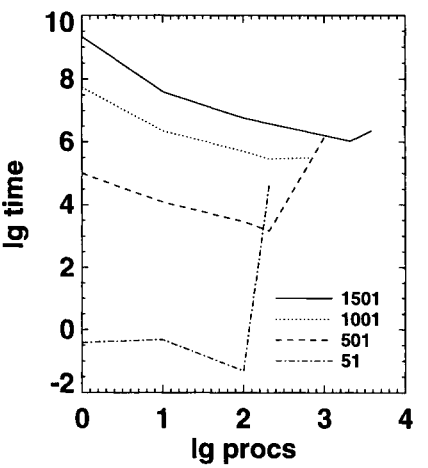


Figure 4. Time taken for 10 steps versus number of processors for different polynomial order of the elements, N. Note that the coordinates are $\log_2$.

In Figure 5a we show the Mflop rate of the code on the SPP-1000 for the various N. The number of floating point operations in the calculation were obtained by running the code on the Cray C90; hence the Mflop rate is a "Cray-equivalent" flop rate. Figure 5b shows the same data on a per processor basis. For $N1501$ and $N1001$ the per node performance increase initially; this is because these values lead to out-of-cache memory requirements. The $N501$ uniprocessor value shows a maximum achieved performance of 42Mflops for the code. For $N1501$ the performance degrades much more slowly than for all the other cases.

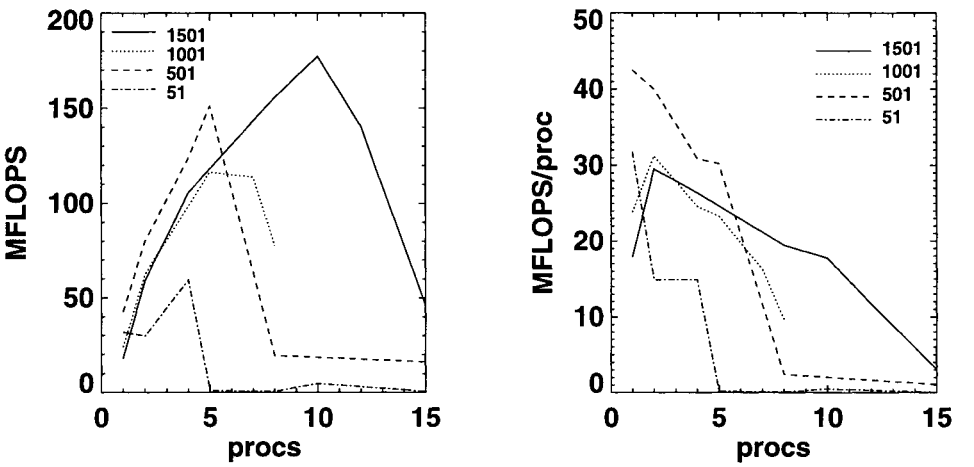


Figure 5. (a) Total Mflop rate versus number of processors, and (b) Mflops/proc versus number of processors, both for different polynomial order of the elements, N.

8. Conclusion

A parallel spectral-element code for compressible hydrodynamics has been constructed, and tested on the Convex SPP-1000. In order to simulate 3D performance high-order elements were chosen such that $N \sim N_x N_y N_z$. Typically, for 3D applications $N_x N_y N_z \sim 10^3$ to 15^3. For large N we find that the SPP-1000 begins to have reasonable scaling. The spawning of the threads was not found to be an issue for large N. Clearly the choice of very large N for a one-dimensional problem is artificial and possibly misleading; with the completion of the construction of the multidimensional code the more relevant scalings will be explored in detail.

9. Acknowledgments

Many thanks to my colleagues Clark Mobarry, Daniel Saverese, and Phil Merkey for valuable insights. This work is supported by NASA High Performance Computing and Communications Program at Goddard Space Flight Center.

REFERENCES

1. Patera, A. T., J. Comput. Phys. **54**, 468 (1984).
2. Crawford, C., Evangelinos, C., Newman, D., and Karniadakis, G., Parallel benchmark simulations of turbulence in complex geometries, in *Parallel CFD 95*, 1995.
3. Giannakouros, J. and Karniadakis, G., Int. J. Num. Meth. Fluids **14**, 707 (1992).
4. Sidilkover, D. and Karniadakis, G. E., J. Comput. Phys. **107**, 10 (1993).
5. Giannakouros, J. and Karniadakis, G., J. Comput. Phys. **115**, 65 (1994).
6. Cai, W., Gottlieb, D., and Harten, A., Cell averaging Chebyshev methods for hyperbolic problems, Technical Report 90-27, ICASE, NASA Langley Research Center, 1990.
7. Deane, A., Zalesak, S., and Spicer, D., 3 D compressible hydrodynamics using Flux Corrected Transport on message passing parallel computers, in *High Performance Computing, 1995*, edited by Tentner, A., pages 128–133, 1995.
8. Boris, J. P. and Book, D. L., J. Comput. Phys. **11**, 38 (1973).
9. Zalesak, S., J. Comput. Phys. **31**, 335 (1979).
10. Zalesak, S., J. Comput. Phys. **40**, 497 (1981).
11. Sod, G., J. Comput. Phys. **27**, 1 (1978).
12. Sterling, T., Saverese, D., Merkey, P., and Gardner, J., An initial evaluation of the Convex SPP-1000 for Earth and Space Science applications, in *Proceedings of the First IEEE Symposium on High-Performance Computer Architecture*, pages 176–185, 1995.
13. Sterling, T., Saverese, D., Merkey, P., and Olson, K., An empirical evaluation of the Convex SPP-1000 hierarchical shared memory system, in *PACT 95*, 1995.

This Page Intentionally Left Blank

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

LARGE SCALE SIMULATIONS OF FLOWS ABOUT A SPACE PLANE USING NWT

Kisa MATSUSHIMA[a] and Susumu TAKANASHI[b]

[a] HPC Systems Engineering Div., Makuhari Systems Laboratory, FUJITSU Ltd.
Nakase 1-9-3, Mihama-ku, Chiba-shi, Chiba, 261, Japan

[b] Aircraft Aerodynamics Division, National Aerospace Laboratory,
Jindaiji-higashi-machi 7-44-1, choufu-shi, Tokyo, 182, Japan

Parallel computation of flowfield about a Space Plane has been conducted on the NWT at the National Aerospace Laboratory (NAL) in Japan. Some of the advantages of large scale simulation are examined from the viewpoint of practical aerodynamic analysis for developing the Space Plane. Through a grid refinement study, it is found that the fine grid can provide fairly improved resolution of flow phenomena and more accurate aerodynamic characteristics. Large scale simulation with fine grid distribution is worth doing for predicting precise flow physics such as boundary layer separation and vortical phenomena.

1. INTRODUCTION

The NAL realized a high performance parallel computer system with distributed memory called NWT in 1993 as the result of the joint research with FUJITSU. The NWT consists of 140 processing elements (PEs). Each individual PE is a vector computer, just the same as FUJITSU VP400, and communicates with the other PEs through a crossbar network. The NWT can provide up to 35 GBytes of memory and up to 236 GFLOPs of computing speed [1,2]. Such high performance has enabled us to carry out large scale Navier-Stokes simulations with a great number of grid points. Authors are interested in the effect of fine grid distribution from the viewpoint of application of CFD to aerodynamic analysis of realistic aircraft, since we have been involved in CFD simulation on the project to develop a Space Plane at the NAL.

This CFD activity started with parametric study. To estimate global aerodynamic forces for plotting aerodynamic characteristic curves, a lot of flows about the Space Plane have been simulated varying the Mach number and the angles of attack, yaw, and rotation. CL, CD and CY obtained from the computations agree with wind tunnel experiment data in the first two digits. In addition, computational results are visualized to do the global survey of flow. Computation mentioned above might be sufficient for understanding overall aerodynamic quality and tendency of a flowfield. However, CFD researchers are hoping to let CFD more useful. Then we have been concerned with what causes the discrepancy in the

third digit of CL and CD values and interested in analysis of physical mechanism of flow phenomena by CFD. We expect the grid size (the number of grid points) is one of important factors and then plan to undertake large scale computation using a great number of grid points on the NWT.

2. NUMERICAL METHOD

2.1 Algorithm to discretize equations

Basic equations to describe a flowfield are the Reynolds averaged thin-layer Navier-Stokes equations. The numerical algorithm is a finite difference method with implicit time integration. In space, Roe's flux difference splitting with the third order MUSCL interpolation is adopted for the convective term and the second order central differencing is for the diffusion term. In time, the LU-ADI method is applied [3]. To evaluate turbulence eddy viscosity, we use the modified Baldwin-Lomax model proposed by Degani and Shiff in their computation of vortical flow at a high angle of attack [4].

2.2 Parallel implementation

Parallel implementation of the Navier-Stokes solver is performed by a simple version of domain decomposition technique, which makes good use of the attribute of replicated local variables of NWT-FORTRAN. Simulation is conducted using 1 to 16 PEs. The number of used PEs depends on the number of grid points. When simulating with 1.1 million grid points, 2 or 4 PEs are used and when simulation with 7.5 million points, 8 or 16 PEs are used. The CPU time it takes to calculate a flowfield at the next time step is about 5.2 μsec per one grid point when simulating with the original solver on single PE, which has no parallel implementation. As for parallel computation, it is 2.7 μsec when 2 PEs are used, 1.4 μsec when 4 PEs are used and 0.36 μsec when 16 PEs are used. In other words, the efficiency of the number effect of PEs is almost 100%, so far.

2.3 Grid system

Two grid systems are prepared for numerical simulations. The first grid is prepared to investigate global aerodynamic characteristics. It contains 1.1 million points in space around a Space Plane and requires about 280 MB of memory storage for computation. Before the installation of the NWT, the simulation was performed with a half grid system in order to save time and memory. The grid system around the whole Space Plane model containing 1.1 million grid points was bisected at the symmetrical plane of the model to be the half system. The half system was used with the assumption that a flowfield must be symmetric.

We prepare a finer grid that contains 7.5 million points for a grid refinement study, aiming to improve the accuracy of local and microscopic aerodynamics of computational results. The second grid requires about 1.8 GB of memory.

The differences between these two grid systems are the distribution along ξ (streamwise) and ζ (normal to the Space Plane surface) directions, and the compression rate of grid spacing inside a boundary layer. The ξ direction is aligning with the longitudinal axis of the Space Plane model and the ζ is from the model

surface to the far field boundary. Along the ξ direction, almost every grid interval of the fine grid is as half as the corresponding grid interval of the coarse one, along the ζ direction, each grid interval of the fine is about one third of that of the coarse. Inside the boundary layer, the fine holds four times as many grid points as the coarse has. The number of grid points of each direction is 110 x 200 x 49 (ξ x η x ζ) in the coarse grid and 270 x 200 x 129 in the fine grid. In the following sections and figures, the first grid system is called 'coarse grid' and the second is called 'fine grid' for convenience.

3. COMPUTATION TO ESTIMATE GLOBAL AERODYNAMICS

Simulations using the coarse grid were conducted on NWT (a vector parallel machine) as well as VP2600 (a vector machine) [5,6]. Figure 1 shows the front and perspective views of emerging and growing of leading edge vortices visualized from one of the computational results for parametric study. The Space Plane is flying at a transonic speed with the angles of attack and yaw. Because of the angles of yaw, the flowfield is unsymmetric. The side wind is blowing from the right side of the figure. Three dimensional traces of stream lines expose many interactions and the development of separation vortices. Interactions between a vortex and vortex, a vortex and side wind, a vortex and the Space Plane's surface, and a vortex and the tail wing. Figure 1 suggests that there possibly occur interesting interactions and complicated vortical phenomena far form the wall of the model where grid distribution is not well considered and the grid spacing there is sometimes too big to resolve the flow physics.

In Figure 2, CL and CD curves are presented as a function of an angle of attack at M_∞=0.9 and 1.5. Dashed lines are denoting data measured by the wind tunnel experiment at the NAL [7], and circles are presenting computational results. The global properties such as CL and CD of computational results are in good agreement with those of experimental results on the plotting scale of Figure 2, while there is discrepancy between experiment and simulation in predicting microscopic aerodynamics [5-7].

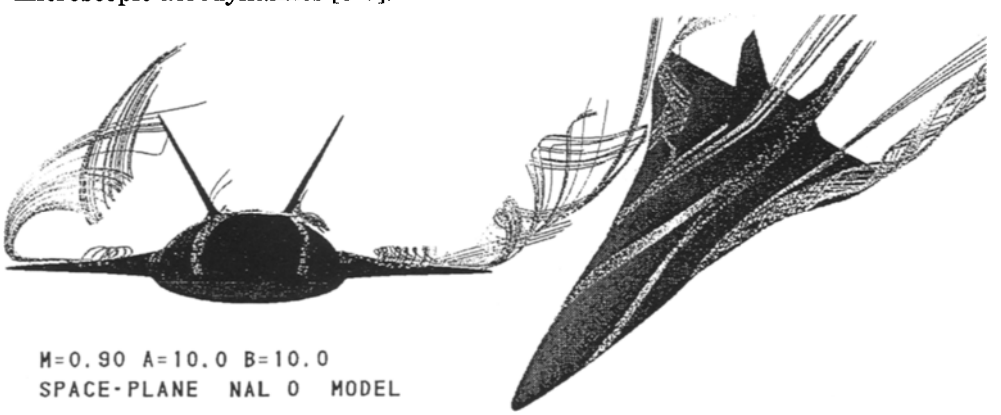


Figure 1 Stream Lines of Leading Edge Vortices ($\mathbf{M_\infty = 0.9}$, $\alpha = 10°$, $\beta = 10°$).

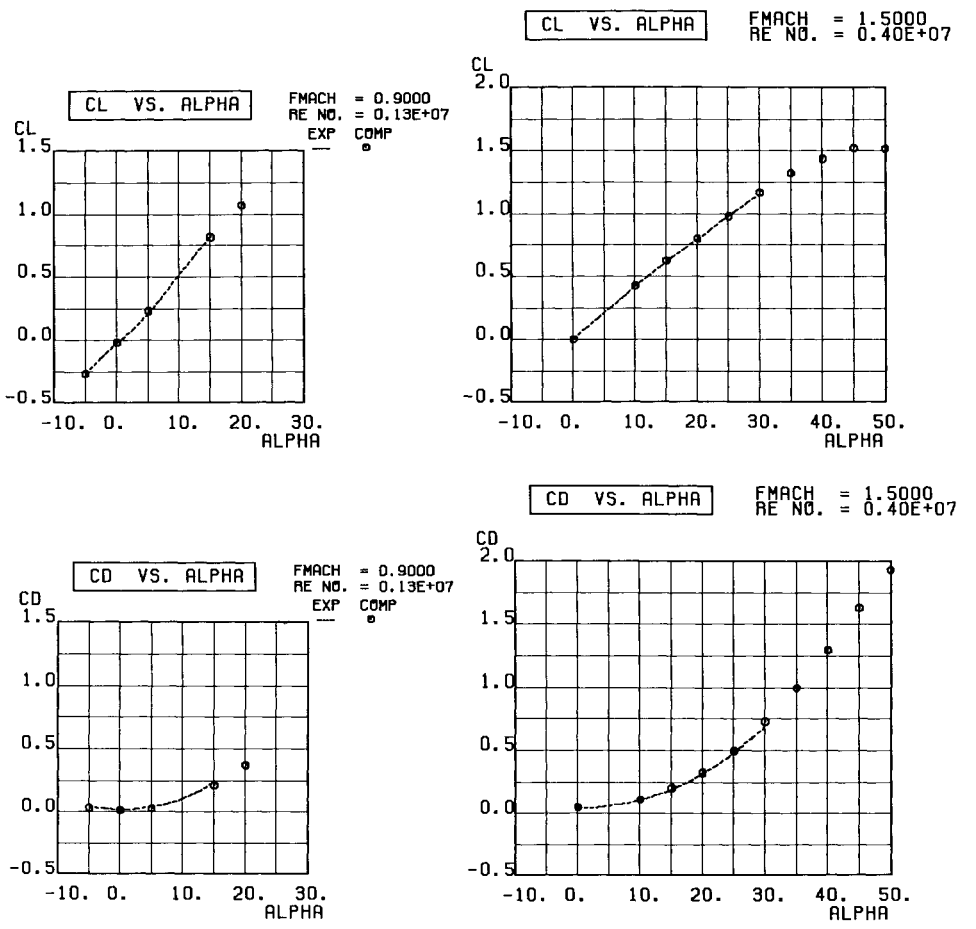


Figure 2 Lift and Drag Coefficients at $M_\infty = 0.9$ and $M_\infty = 1.5$.

4. COMPUTATION FOR GRID REFINEMENT STUDY

Computation with the fine grid is being carried out using 8 or 16 PEs on NWT. In this study, we choose the free stream conditions as $M_\infty = 1.5$, and $\alpha = 15°$. In figures 3-5, the simulation results using both grids are compared. They show the density contour map and the surface Cp distribution in a cross flow sectional plane at several stations of ξ direction. x/L means the station; x is the distance from the nose tip of the model and the L is the whole length of it. Cp distribution measured by the NAL's wind tunnel experiment is also plotted just for reference [7]. Circles are denoting experimental Cp data while solid and broken lines are Cp distributions

predicted by computation These figures indicate that the present computation using the fine grid can give an improved result in capturing vortex phenomena and predicting pressure values.

4.1 Cross flow phenomena at x/L=0.60

In figure 3, on the upper surface of the body, two vortices definitely appear when the fine grid is used. They are caused by the leading edge separation somewhere on the former surface of the Space Plane. They smear out when the coarse grid is used. On the lower side of the wing-body juncture, concentration of contour lines lying separately from the surface is observed only in the density contour map of the fine grid. We call it a detached boundary layer, not a separated boundary layer, because a separated boundary layer usually forms a vortex and doesn't remain to be such a shear layer as seen in the density contour map. In the narrow region surrounded by the lower surface and the detached boundary layer, a flat anti-clockwise vortex is observed through other visualization. These two facts make the computed Cp distribution using the fine grid in better agreement with the experimental one on either surface.

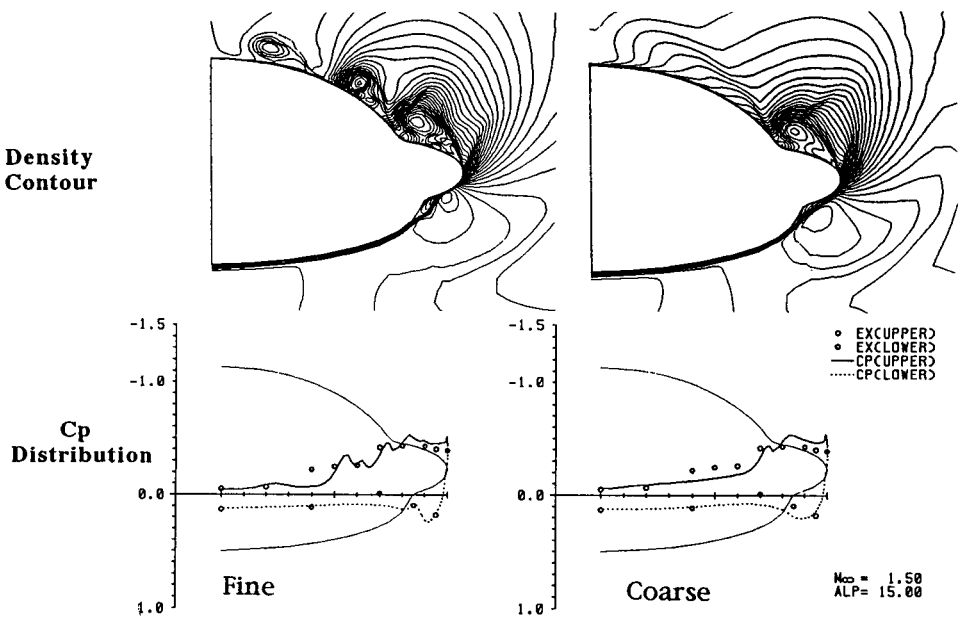


Figure 3 Comparison of Computational Results (M_∞= 1.5, α = 15°, x/L = 0.60).

4.2 Cross flow phenomena at x/L=0.83

The density maps of Figure 4 indicate three distinguished differences between two computations. One of three is a vortex located near the symmetrical line of the body; the others are detached boundary layers on the upper and lower sides of the wing-body juncture. Flow is moving to the ξ direction circulating inside the flat

region surrounded by the Space Plane surface and detached boundary layers. On the upper surface of Figure 4, the accuracy of predicted Cp values with the fine grid is improved at the wing-body juncture. The improvement is explained by the fact that the fine grid can resolve complicated phenomena of boundary layer and vortex interaction, such as detached boundary layer in the density contour map. Along the lower surface, Cp distribution obtained by computation disagree with that by the experiment. The reason of the discrepancy lies on the existence of a big sting supporting the Space Plane model for the wind tunnel experiment.

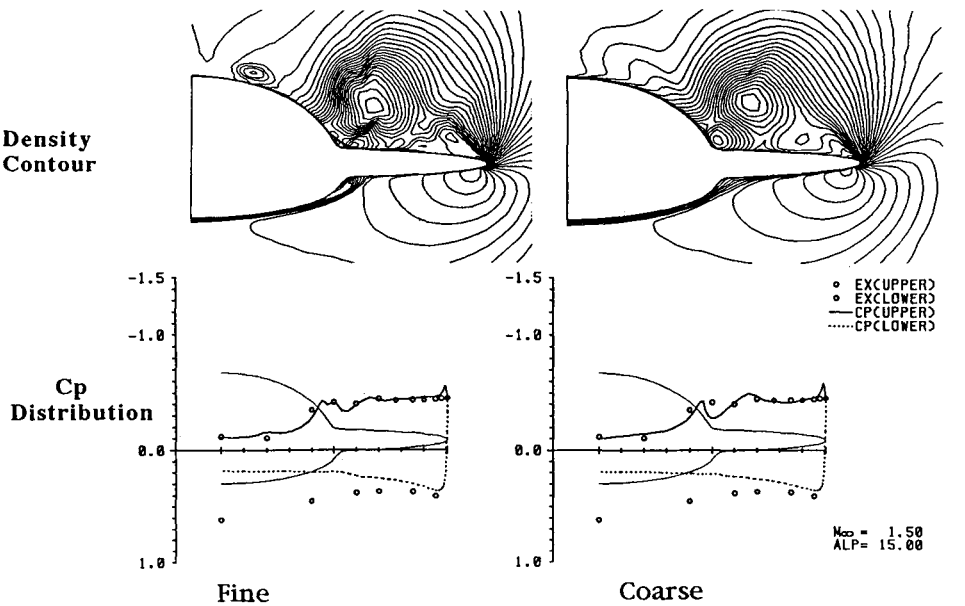


Figure 4 Comparison of Computational Results ($M_\infty = 1.5$, $\alpha = 15°$, $x/L = 0.83$).

4.3 Cross flow phenomena at x/L=0.91

In figure 5, boundary layer behavior at two locations appears differently on two computations using the coarse and fine grids. Only the fine grid can resolve the detached boundary layers on both sides of the body-wing juncture. In respect of vortical flow phenomena, there is no big difference between two computations. On the upper surface of this cross section, both of the Cp distribution curves of two computations show good agreement with the experimental one. The discrepancy between experimental and computational data on the lower surface is explained by the same reason as that at the station of x/L=0.83, the sting for the experiment. At this station, flow looks complicated with some vortices but stable. Flow doesn't seem to critically respond to small disturbances because of the positive effect of delta-shape main wings and v-shape tail wings.

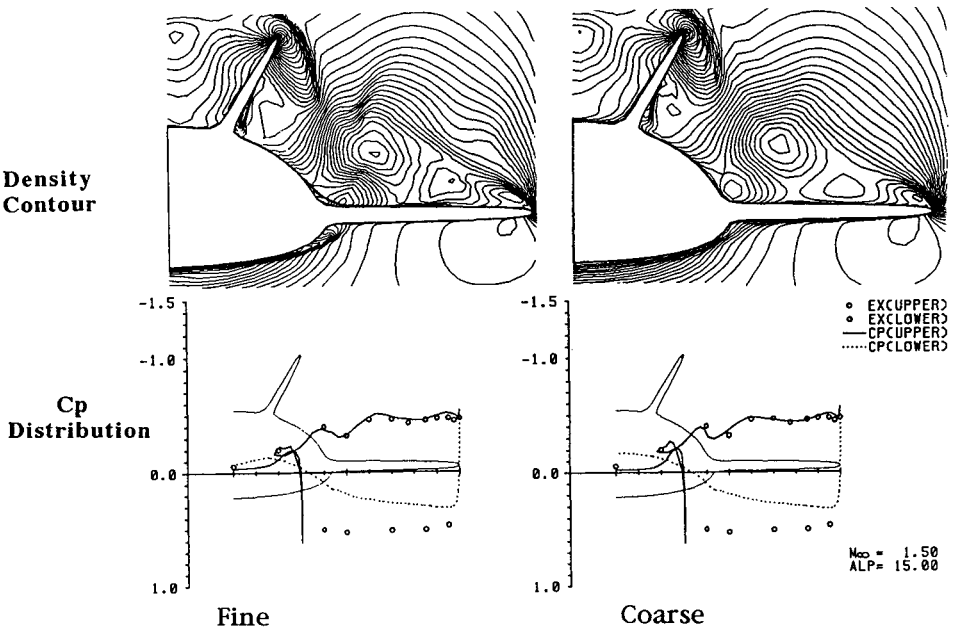


Figure 5 Comparison of Computational Results (M_∞= 1.5, α = 15°, x/L = 0.91).

5. CONCLUSION

Navier-Stokes simulation of flows about the Space Plane has been carried out using two grid distributions. They differ in the grid spacing in the ξ (stream wise) and the ζ (normal to the surface) directions, and the number of grid points in a boundary layer. The finer grid distribution improves resolution of flow physics especially on behavior of a boundary layer and vortices, and on rotating flow in a boundary layer. Higher resolution yields more accurate estimation of aerodynamic characteristics of the Space Plane. However, there are still some differences from experimental results in the present simulation.

We admit other factors that influence the computational accuracy, such as turbulence models have to be investigated. We still dare to conclude grid spacing has to be decided in accordance with the physical scale of a phenomenon to be simulated Grid distribution should be finer than half of the physical scale.

This motivates us to challenge larger scale computation for predicting more accurate flow physics using finer grid distribution of which grid spacing in η (circumference on cross sections) direction is as half as the present grid.

REFERENCES

[1] Miyoshi, H., et al., Development and Achievement of NAL Numerical Wind

Tunnel for CFD Computations, IEEE Supercomputing' 94, (1994).
[2] Iwamiya, T. and Fukuda, M., Numerical Wind Tunnel of National Aerospace Laboratory, WBPE, (1993).
[3] Fujii, K. and Obayashi, S., Navier-Stokes Simulation of Transonic Flows over a Practical Wing Configuration, AIAA Journal, Vol.25, No.3, 1987.
[4] Degani, D. and Schiff, L. B., Computation of Supersonic Viscous Flow Around Pointed Bodies at Large Incidence, AIAA Paper, 83-0034.
[5] Matsushima, K., et al., Navier-Stokes Computations of the Supersonic Flows about a Space Plane, AIAA Paper 89-3402CP, (1989).
[6] Matsushima, K. and Takanashi, S., Navier-Stokes Simulations of Transonic Flows about a Space Plane, AIAA Paper 94-1864CP, (1994).
[7] Sato, M., et al., Aerodynamic Characteristics of a Space Plane at NAL's Transonic/Supersonic Wind Tunnel, 19th annual Meeting of JSAA, (1988). (In Japanese)

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

LARGE SCALE NAVIER–STOKES AERODYNAMIC SIMULATIONS OF COMPLETE FIGHTER AIRCRAFT ON THE INTEL PARAGON MPP

J. Vadyak and G. D. Shrewsbury
Lockheed–Martin Skunk Works, Palmdale, California, USA

G. Montry
Southwest Software, Albuquerque, New Mexico, USA

V. Jackson, A. Bessey, G. Henry, E. Kushner, and T. Phung
Intel Scalable Systems Division, Beaverton, Oregon, USA

1.0 SCOPE

This paper describes the development of a general three dimensional multiple grid zone Navier–Stokes flowfield simulation program (ENS3D–MPP) designed for efficient execution on the Intel Paragon Massively Parallel Processor (MPP) supercomputer, and the subsequent application of this method to the prediction of steady viscous flowflelds about complete modern fighter aircraft using very large scale viscous computational grids. The present paper focuses on porting strategies and timing results for viscous simulations ranging from roughly 0.5 million to 30 million mesh points with the number of compute nodes varying between 30 and 1024. Some preliminary results using the lastest Paragon MP compute node are also discussed.

2.0 DESCRIPTION OF GRID GENERATION AND FLOW SIMULATION METHODS

2.1 Background of Grid Generation Methods

A family of computer analysis programs have been developed at Lockheed–Martin for calculating steady, or unsteady, three–dimensional viscous turbulent flowfields. These programs can compute the flowfield for individual aircraft components or complete aircraft configurations over a large range of flow conditions. The flowfield solution for a given configuration is determined on a structured body–fitted, three–dimensional, curvilinear computational mesh. The computational mesh for each different configuration is determined by a separate grid generation computer program. A variety of grid generation programs can be used in conjunction with the flow analysis codes. Most of the existing mesh generation programs rely on numerical grid generation techniques which are based on solving a system of coupled elliptic or parabolic partial differential equations. Isolated component geometries are typically analyzed using a single block grid approach. Multi–component configurations are typically analyzed using a multi–block grid approach where the global computational grid is comprised of a series of sub–grids which are patched together along common interface boundaries.

2.2 Background of Flow Analysis Methods

Once the computational grid is generated, the flowfield solution can be obtained using a version of the Lockheed–developed ENS3D (Elliptical Navier–Stokes in 3–Dimensions) flowfield simulation program by solving either the full three–dimensional Reynolds–averaged Navier–Stokes equations, the thin–shear–layer Navier–Stokes equations, or the Euler equations.

The governing equations are cast in strong conservation–law form to admit solutions in which shocks are captured. Second–order differencing is used in computing the metric parameters which map the physical domain to the computational domain. A time–marching, fully–implicit, approximate factorization scheme (Ref. 1) is used for solution of the finite–difference equations. This produces a system of block tridiagonal equations to be solved for each time step. Either

steady–state or time accurate solutions can be obtained, with second–order or fourth–order spatial accuracy and first– or second–order temporal accuracy. The convective (inviscid) terms in the governing equations can be differenced using either central differencing or TVD upwind differencing using an extension to three–dimensional viscous flow of Harten's method (Ref. 2). The upwind differencing option considers the range of influence and domain of dependence at a solution mesh point by examining the local characteristic fields. The viscous diffusion terms employ central differencing. The algorithm includes the grid speed terms in the contravarient velocity calculations, thereby permitting the computation of unsteady flows with a time–varying grid. A solution adaptive grid method capability is also present.

Although the interior points are updated implicitly, an explicit boundary condition treatment is employed which allows for the ready adaption of the program to new configurations. To aid convergence, non–reflecting subsonic outflow boundary conditions are employed along with a spatially varying time step for steady–flow solution cases. For the central difference option, the algorithm can use either a constant coefficient artificial dissipation model or a variable coefficient model where the coefficient's magnitude is based on the local pressure gradient. For the upwind differencing option, the algorithm is naturally dissipative. Laminar viscosity is computed for viscous cases using Sutherland's law or, for a mixture of gases, using parametric curve fits as a function of temperature while employing Wilke's law (Ref. 3) to obtain the effective mixture molecular viscosity. For turbulent viscous flows, the effective eddy viscosity can be computed using either the Baldwin–Lomax two–layer algebraic turbulence model (Ref. 4), the Johnson–King one–half equation model (Ref. 5), or the k–e two–equation transport model (Ref. 6). The k–e model requires the solution of 2 additional partial differential equations.

Versions of the ENS3D flow analysis algorithm were developed that can account for real gas effects to permit high speed flow simulations or simulations involving gas mixtures. Three thermochemical models were incorporated with varying degrees of complexity and computer resource requirements.

Another version of ENS3D was developed to treat aeroelastic effects by simultaneously solving the Navier–Stokes equations and a structural dynamics model (Ref. 12). This version, ENS3DAE, was developed under funding from the U.S. Air Force Wright Laboratory and is in the public domain. The ENS3DAE code is written to handle arbitrary block grid topologies and has been recently upgraded with additional propulsion boundary conditions options, real gas effects, turbulence model options, and convergence acceleration schemes. Aeroelastic analyses are obtained by coupling a set of structural dynamics equations of motion to the aerodynamic solution. This is accomplished by using pressures computed by the aerodynamic analysis as forcing functions for the structural dynamic equations. The structural equations are solved using an explicit predictor–corrector scheme to ultimately obtain structural deflections. A fully coupled, time–accurate aeroelastic solution is thereby obtained.

2.3 ENS3D Execution on Supercomputers

In an effort to reduce the required computer execution time, vectorized and vectorized/parallelized versions of ENS3D have been developed to execute on shared memory vector/parallel supercomputers such as the Cray Y–MP, and Cray C–90. In addition, versions of ENS3D have been developed for various Massively Parallel Processor (MPP) platforms such as the Masspar MP–2216, the Thinking Machines Incorporated CM–5, the Kendall Square Research KSR–1 and KSR–2, the Intel Paragon and iPSC–860. On the Cray Y–MP vector/parallel machines, sustained processing rates of about 180 to 230 megaflops per CPU are

typical. ENS3D exhibits over 98% parallelism on the these platforms. At high vector lengths with a dedicated machine sustained processing rates of about 8 to 9 gigaflops are obtained on a Cray C–90.

3.0 PORTING TO THE INTEL PARAGON MPP

The ENS3D flow simulation code was ported to Intel distributed memory Paragon MPP. Parallelization on the MPP was obtained using extensive explicit message passing. Two main MPP versions were developed; namely the scalable version and the superscalable version. The Intel NX message passing library was used for the Paragon MPP coding. The Intel MPP coding uses double precision arithmetic to produce 64 bit words. Synchronous message passing is used currently, but a MPP version using asynchronous message passsing is also under development.

The scalable version uses a global/local time–stepping procedure where one grid zone is processed at a time in the compute node partition on the MPP, as shown in Figure 1. When the specified number of local iterations or time steps has been completed for the given zone, another grid zone is mapped into the compute node partition. This is performed for all grid zones in the global mesh, after which the next global time step is taken, and the overall procedure is repeated for the specified number of global cycles.

To obtain the solution within a grid zone for a time step a one–dimensional domain decomposition is employed by assigning each K or a group of K planes in the grid block (I,J,K) space to a given processor on the MPP, as shown in Figure 2. Grid and flow property data in the I and J directions for this grouping of K planes are contained in the memory of the assigned node. Data in the K direction not contained with the local node's memory and needed for calculation is passed by using a system of ghost planes. When the sweeps in the I and J directions are completed, a data transpose technique is used to allow sweeping in the K direction. A final transpose then ensues before the next time step.

The code allows arbitrary block grid topologies to be analyzed. Block to block interfaces may be of like or different curvilinear grid families with arbitrary ordering and direction. Message passing is used to produce interblock connectivity. The scalable version is limited to using a maximum number of processors equal to the smallest interior cross plane dimension of any grid block.

The superscalable version creates a pseudo–compute node partition for each grid zone, and maps the entire global mesh topology onto the MPP as shown in Figure 3. A MAP library (Ref. 7), developed at NASA–Ames, is used to communicate between the pseudo partitions. This code version allows all the grid zones to be integrated in time concurrently. This allows for operation on hundreds or thousands of compute nodes and greatly reduces wall clock execution time and increases scalability. Because grid and solution data does not have to be rolled in and out of a single compute node partition as in the scalable version, the solution time per given grid zone is also reduced. To minimize the required I/O time, nodes within each respective zone partition are used to perform I/O for that grid zone only. Optimum mapping occurs when one processor is used per K–plane per grid zone.

4.0 APPLICATIONS

4.1 General Applications of ENS3D

ENS3D has been used to analyze numerous aircraft components and complete aircraft

configurations. Inlets, nozzles, wings, forebodies, and wing/body component configurations have been analyzed. Also airframe viscous analyses have been performed for the YF–22, F–16, F–15, F–117A, ARPA ASTOVL, Navy AX, SR–71, U–2, NASP, SSTO, V–22, FDL–5A, and FDL–5B vehicles among others including many classified configurations. Some of these applications are discussed in Refs. 8 to 13. The application in Ref. 13 is for the V–22 Osprey tiltrotor vehicle using a multizonal global grid with the flow simulation being performed on the Intel Paragon and is being sponsored by NASA under the Computational AeroSciences Program.

4.2 Large Scale Fighter Aircraft Simulations on the Intel Paragon MPP

Using the Intel Paragon superscalable version of the code, very large scale 3–D Navier–Stokes viscous flow simulations have been performed. Dense mesh solutions on global grids ranging up to nearly 30 million points were performed for the F–117A aircraft and a generic Advanced Short Takeoff and Vertical Landing (ASTOVL) aircraft. The largest of these simulations were performed on the 512 GP–node Intel Paragon at the Caltech Center for Advanced Computing Research in Pasadena, California and on the 1024 MP–node Intel Paragon at the Oak Ridge National Laboratory in Oak Ridge, Tennessee.

Navier–Stokes flow simulations were performed on semi–span and full–span global meshes which were comprised of 4 to 8 grid zones with common interfaces being used at the zonal boundaries. A side view of a typical block grid topology used is given in Figure 4. The mesh consists of a series of sheared Cartesian H–H zones with grid clustering being used to resolve the boundary layers. The H–H block grids were generated using the Complete Aircraft Mesh Program (Ref. 12).

Figure 5 illustrates the generic ASTOVL vehicle which was simulated along with computed particle paths showing vortex formation emanating from the leading edges of the canards and the wings. This case corresponds to the conditions of a free–stream Mach number of 0.17 and 30 degrees angle of attack.

Three–dimensional viscous Navier–Stokes flow simulations were performed on the Intel Paragon for the complete F–117A stealth fighter and the ASTOVL fighter for varying grid sizes. Resulting computation times, excluding times for input and output, are presented in Tables 1 through 4. Tables 1 and 2 present timing results for the ASTOVL vehicle shown in Figure 5. Tables 3 and 4 present timing results for the F–117A simulations at transonic Mach number at incidence with 0 (semi–span) and nonzero (full–span) sideslip angles.

Table 1 gives computation timing results for a global mesh consisting of 5 grid zones, each zone having 100 axial stations, 32 spanwise stations, and 32 normal stations, thereby giving a mesh of 512,000 grid points. Shown in this table are timing results for 30, 50, 75, abnd 150 compute nodes. Wall clock times to compute a Global Time Step (GTS) and to compute the final solution are given. The time per Global Time Step (GTS) is defined as the wall clock time needed to advance the solution for all points in all grid zones with 5 local time steps per grid zone being performed. The total time is the wall clock time needed to advance the soltion 200 global time steps or 1000 effective cycles through the entire mesh. At each grid point, 5 solution variables are computed which are comprised of the density, 3 velocity components, and the internal energy. Table 1 shows good scalability using the Case A–1 simulation on 30 compute nodes as the base. Actual and theoretical linear scalings are presented in the rightmost two columns.

Table 2 presents compute timings for a 8x100x62x62 grid with 3,075,200 total mesh points being executed on 120 through 480 compute nodes. These cases were also for the ASTOVL

vehicle. With this larger grid size, nearly perfect linear scaling is observed with the wall clock time for a 480 node execution being 1.508 hours.

Table 3 presents Navier–Stokes computation times for the complete F–117A vehicle on two meshes. The first semi–span mesh had 4 zones with each zone having 100 axial stations and cross plane grid dimensions of 102x102. This gives a total of 4,161,600 mesh points. The second mesh was used for full–span simulations and had 8 grid zones each being 100x102x102. This mesh had 8,323,200 points and is exactly twice as large as the first mesh. Computation times are presented for 400 compute nodes for the first mesh, and for 800 and 400 nodes for the second mesh. Doubling the mesh size and doubling the number of compute nodes produces nearly linear scaling. Doubling the mesh size and retaining the number of nodes at 400 results in super–linear scaling in that a 1.84 factor increase in wall clock time was observed with theoretical linear scaling indicating that a factor of 2.0 increase would have been required.

Table 4 presents the computation times for two different sizes for additional F–117A flow simulations. The first mesh consisted of 8 grid zones with each zone having 80 axial stations and cross plane dimensions of 130x130, giving a total of 10,816,000 points. The second mesh is similar except having 160 axial stations per zone, thereby having 21,632,000 mesh points. This is exactly twice as large as the first mesh. The Case D–2 mesh solution requires the calculation of 5.33X10**8 final solution variables per global time step. Once again, a super–linear scaling was noted in comparing the wall clock times for Cases D–1 and D–2. The 21,632,000 mesh case required only 1.56 times the wall clock time of the 10,816,000 mesh case instead of a linear scaling factor of 2.0. Case D–3 presents results for a mesh with 8x220x130x130 or 29,744,000 points. In this case, a scaling ratio of 2.25 was observed using Case D–1 as the base. Linear scaling gives a factor of 2.75.

Tables 1 through 4 presented computation timing results excluding Input/Output (I/O) times. The only signicant I/O is performed at the beginning and end of the execution to load the grid data and to write the resulting flow solution data, respectively. Table 5 presents elasped times for program output for the 3,075,200 mesh point case whose comptutation times are given in Table 2. Results are given in Table 5 using the Paragon Unix File System (UFS) and the Paragon Parallel File System (PFS) for varying number of compute nodes. As discussed earlier, each grid zone employs its own I/O node set. For the 120 compute node case, the Parallel File System required only about 16% of the wall clock time required by the Unix File System to transfer the same amount of flow solution output data.

4.3 Tuning for the MP node Paragon

Recently, a second–generation Paragon system (Paragon MP) has become available from Intel. The primary difference between the new system and the first–generation sytem is an additional i860 microprocessor on each node. Thus the Paragon system has 2 i860's per node (one for computation and one for communication) while the Paragon MP system has 3 i860's per node (two for computation and one for communication). Concurrent execution of the 2 computational processors is achieved automatically by the Fortran and C compilers. These compilers detect loops that can be run concurrently and activate a thread of execution on each processor that remain active for the duration of the loop. The limitation to which a second processor can be used is the bandwidth to memory which supports both processors and is limited to 400 MBytes/sec.

For ENS3D, the primary operation that benefits from the additional processor is the solution of the block tridiagonal systems of linear equations. Using a small problem on a small number of

nodes as a test case, it was determined that this operation runs about 20% faster on two processors. Since the block tridiagonal solver consumes about half the time, this represents a 10% reduction in run time. Efforts are now under way to integrate this optimization into the standard version of the code and then run the largest problems on very large partitions. All the results presented in Tables 1 through 4 use only one of the two application processors on the Paragon MP compute node.

Table 1. Timing Results for 32x32 Cross Plane Dimension per Zone

Case	Grid Size	Mesh Points	Compute Nodes	Time per GTS (sec)	Compute Time (hrs)	Ratio to A–1 Actual	Theo
A–1	5x100x32x32	512,000	30	62.40	3.466	1.00	1.00
A–2	5x100x32x32	512,000	50	38.97	2.165	0.62	0.60
A–3	5x100x32x32	512,000	75	28.17	1.565	0.45	0.40
A–4	5x100x32x32	512,000	150	16.87	0.937	0.27	0.20

Table 2. Timing Results for 62x62 Cross Plane Dimension per Zone

Case	Grid Size	Mesh Points	Compute Nodes	Time per GTS (sec)	Compute Time (hrs)	Ratio to B–1 Actual	Theo
B–1	8x100x62x62	3,075,200	120	98.62	5.478	1.00	1.00
B–2	8x100x62x62	3,075,200	160	72.91	4.050	0.74	0.75
B–3	8x100x62x62	3,075,200	240	51.32	2.851	0.52	0.50
B–4	8x100x62x62	3,075,200	480	27.15	1.508	0.27	0.25

Table 3. Timing Results for 102x102 Cross Plane Dimension per Zone

Case	Grid Size	Mesh Points	Compute Nodes	Time per GTS (sec)	Compute Time (hrs)	Ratio to C–1 Actual	Theo
C–1	4x100x102x102	4,161,600	400	42.74	2.374	1.00	1.00
C–2	8x100x102x102	8,323,200	800	44.21	2.456	1.03	1.00
C–3	8x100x102x102	8,323,200	400	78.55	4.363	1.84	2.00

Table 4. Timing Results for 130x130 Cross Plane Dimension per Zone

Case	Grid Size	Mesh Points	Compute Nodes	Time per GTS (sec)	Compute Time (hrs)	Ratio to D–1 Actual	Theo
D–1	8x80x130x130	10,816,000	1024	55.64	3.091	1.00	1.00
D–2	8x160x130x130	21,632,000	1024	86.71	4.817	1.56	2.00
D–3	8x220x130x130	29,744,000	1024	124.97	6.94	2.25	2.75

Table 5. Input/Output Timing Results for 8x100x64x64 Grid Case

Case	Compute Nodes	Time for Output Using PFS (sec)	Time for Output Using UFS (sec)
B–1	120	75.5	470.3
B–2	160	91.1	480.0
B–3	240	103.1	.....
B–4	480	132.1	779.5

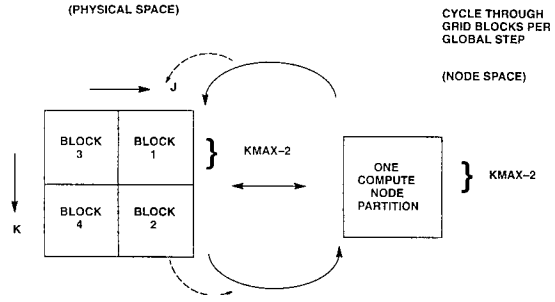


Figure 1. Scalable MPP Version Operation.

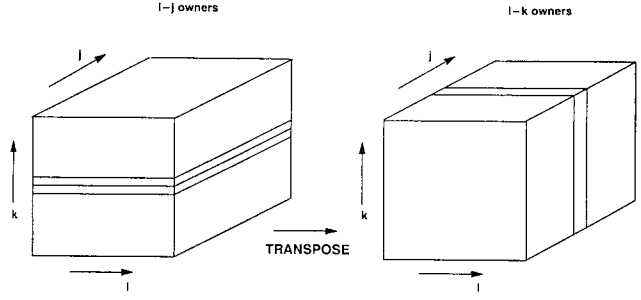


Figure 2. Domain Decomposition of a Grid Zone.

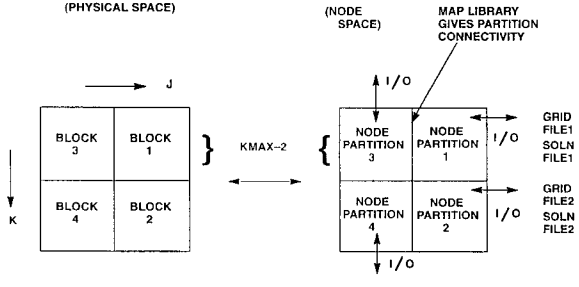


Figure 3. Superscalable MPP Version Operation.

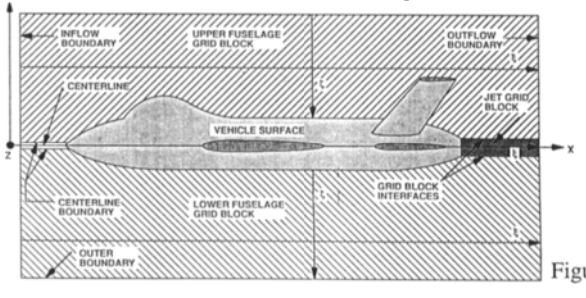


Figure 4. Zonal Volume Grid Topology for Fighter Aircraft Analysis.

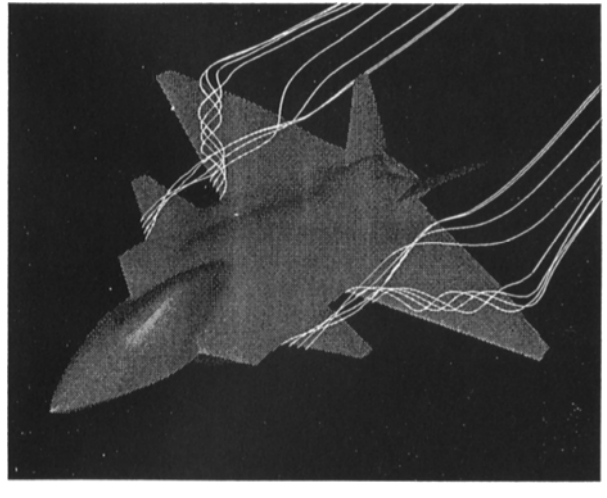


Figure 5. Generic ASTOVL Fighter Showing Vortex Formation at Mach=.17 and 30 Degrees Angle of Attack.

ACKNOWLEDGEMENT

Application of the ENS3D–MPP code to the V–22 vehicle flowfield analysis is being sponsored by NASA–Ames under Contract NAS2–14095. Intel Paragon computing time was supplied by NASA–Ames, the Caltech Center for Advanced Computing Research (CACR), the Aeronautical Systems Center at Wright–Patterson AFB, Intel Scalable Systems Division, and the Oak Ridge National Labratory Center for Industrial Innovation. The authors wish to express their deep appreciation for this support.

REFERENCES

1. Vadyak, J., "Simulation of Diffuser Duct Flowfields Using a Three–Dimensional Euler/ Navier–Stokes Solver," AIAA Paper 86–0310, 1986.
2. Yee, H.C., and A. Harten, "Implicit TVD Schemes for Hyperbolic Conservation Laws in Curvilinear Coordinates," AIAA Journal, Vol. 25, No. 2, 1987.
3. Eckert, E.R.G., and R.M. Drake, "Analysis Of Heat And Mass Transfer," McGraw–Hill, New York, 1972.
4. Baldwin, B.S., and H. Lomax, "Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows," AIAA Paper 78–257, 1978.
5. Johnson, D.A., and King, L.S., "A New Turbulence Closure Model for Attached and Separated Turbulent Boundary Layers," AIAA Journal, Vol. 23, No. 11, 1985.
6. Gorski, I.J., "A New Near–Wall Formulation for the k–e Equations of Turbulence," AIAA Paper 86–0556, 1986.
7. Fineberg, S.A., "The Design of a Flexible Group Mechanism for the Intel Paragon XP/S", Computer Scienecs Corp., NAS, NASA–Ames Research Center, Moffett Field, CA.
8. Vadyak, J., M.J. Smith, and D.M. Schuster, "Navier–Stokes Simulations of Supersonic Fighter Intake Flowfields," AIAA Paper 87–1752, 1987.
9. Vadyak, J., M.J. Smith, and D.M. Schuster, and R. Weed, "Simulations of External Flowfields Using a 3–D Euler/Navier–Stokes Algorithm," AIAA Paper 87–0484, 1987.
10. Vadyak, J., M.J. Smith, and D.M. Schuster, and G. Shrewsbury, "Simulations of Aircraft Component Flowfields Using a Three–Dimensional Navier–Stokes Algorithm," presented at the 3rd International Symposium on Science and Engineering on CRAY Supercomputers, Minneapolis, Minn., Sept 9–11, 1987.
11. Vadyak, J. and Schuster D.M., "Navier–Stokes Simulation of Burst Vortex Flowfields for Fighter Aircraft at High Incidence, "AIAA Journal of Aircraft, Vol. 28, No. 10, October 1991.
12. Schuster, D.M. Vadyak, J., and Atta, E.H. "Static Aeroelastic Analysis of Fighter Aircraft Using a Three–Dimensional Navier–Stokes Algorithm, "AIAA Journal of Aircraft, Vol 27, No. 9, September 1990.
13. Vadyak, J., Shrewsbury, G.D., Narramore, J., and Montry, G., "Navier–Stokes Aerodynamic Simulation of the V–22 Opsrey on the Intel Paragon MPP", Computational AeroSciences (CAS) Workshop held at NASA–Ames Research Center, March 7–9, 1995.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Benchmarking the FLOWer code on different parallel and vector machines. †

C.W. Oosterlee, H. Ritzdorf,

GMD–German National Research Center for Information Technology,
Inst. for Algorithms and Scient. Computing (SCAI), Sankt Augustin, Germany.

H.M. Bleecke and B. Eisfeld,

DLR–German Aerospace Research Establishment,
Inst. for Design Aerodynamics, Braunschweig, Germany.

1 Introduction

POPINDA is a German cooperation project between Daimler Benz Aerospace (DASA), Daimler Benz Aerospace Airbus, DLR, GMD, IBM Scientific Center Heidelberg, and ORCOM. A central project goal is to utilize parallel systems for aerodynamic production codes, and in particular to parallelize 3D industrial compressible Navier-Stokes solvers based on a communications library. At present in POPINDA two parallel Navier-Stokes codes exist: FLOWer, a cell vertex code and NSFLEX, a cell-centered code. The DLR code CEVCATS serves as basis for the cell vertex code, which has been further developed with contributions by Airbus, DASA and DLR. A description of this code is given in [2], [4]. The sequential NSFLEX code, developed at DASA is the basis for the parallel NSFLEX code. Both sequential codes have been successfully parallelized using the high-level GMD Communications Library CLIC ([6]). The library provides subroutines for all communications tasks occurring in 2D and 3D multiblock multigrid applications based on vertex- or cell-centered discretizations. Furthermore, CLIC routines analyze the block structure, classify boundary points topologically, and detect geometrical singularities. Its portability and therefore that of the application codes is assured by an implementation using the message passing interface PARMACS ([1]). A next version of the CLIC library will be based on the message passing interface MPI. With CLIC the aerodynamics codes, consisting of approximately 65000 and 25000 lines of code, have been parallelized within one week, which is no small achievement.
In the present work wall clock times obtained with the parallel FLOWer code are compared with the sequential code. We consider two problems, a 3D Euler flow around a

†This work was supported by the German Federal Ministry for Research and Education under contract nr. IR302A7 (POPINDA Project).

NACA0012-wing and a flow around a wing-body configuration. Both grids are divided into several blocks, so that the performance on a vector computer can be compared with the performance on a parallel machine. Results are presented on the MIMD computers IBM SP2, NEC Cenju–3 and Intel Paragon. Wall clock times are also known for some (parallel) vector machines, such as a Cray C90 with 12 processors, a Cray J90 with 8 processors and an NEC SX3.

2 The FLOWer code

Three-dimensional compressible Euler and Navier-Stokes flow problems in general domains can be solved by the FLOWer code. The equations are discretized on block-structured curvilinear grids with cell vertex based finite volumes. In every control volume the equations of mass, momentum and energy are set up as a difference of fluxes ([4]). The fluxes are computed with a second order central difference scheme and addition of artificial dissipation. Steady flows are being solved using an unsteady algorithm based on an explicit multistage Runge-Kutta scheme. The code employs a multigrid algorithm to accelerate convergence. Due to the explicit character of the algorithm and the block-structured data management FLOWer is well suited for efficient use of parallel machines. The usual way of parallelizing this type of code is called grid partitioning ([5], [2]). The global grid is split into blocks, each of which is logically rectangular. Blocks can be put together almost arbitrarily, so that complicated geometries can be covered. The application is parallelized by assigning each block to a different process. Along the interior block boundaries the grid is stored with some overlap. Keeping the values in overlap regions up-to-date on all multigrid levels requires communication between the nodes.

3 The Communications Library CLIC

A portable interface can be defined at various levels of abstraction. One possibility is to identify high–level communication patterns in a class of applications, and to design subroutines of a central library, which handle those communication tasks. The GMD Communications Library CLIC (Communications Library for Industrial Codes) is such a high-level library. The target applications are PDE solvers on regular or block-structured grids, as they result from finite difference or finite volume discretizations. In particular, the library supports parallel multigrid applications. For this class of applications it turns out that, although the numerics differ widely, the communication sections are quite similar in many programs, depending only on the underlying problem geometry. If a library covers all communication requirements of an application class, the user programs themselves do not contain any explicit message passing, and are thus independent of any vendor–specific interface. Only the library has to be implemented on the different platforms. Programming for multiblock geometries is then as easy as for a single cube, and

algorithmic development on an application code and CLIC development can be done simultaneously. The CLIC user interface provides the application program with all required information about the problem geometry.
CLIC is based on PARMACS 6.0 and is designed for a host–node (master–slave) model. CLIC routines in the host program read in the description of the block-structured grid, create node processes, distribute blocks in a load–balanced way to allocated node processors, and input parameters to node processes. Another routine reads grid coordinates and sends them to corresponding node processes. Each node process executes the node program, which is similar to the sequential user program. Of course, a node process receives the information and grid coordinates only for the blocks for which the node process performs grid computations. Library routines also analyze the block structure; i.e. for each segment edge and segment point, the adjoining blocks and the number of coinciding grid cells are determined and the edge or point is classified topologically. If it is part of a physical boundary, the physical boundary conditions of all adjoining blocks are determined. In addition, grid coordinates can be examined and geometrical singularities, such as block faces which collapse to a single point, can be detected. All these data can be interrogated and may be used in the user program; for example in the discretization of irregular grid points or physical boundary points. These data may be important for the user program, however, they are essential for CLIC to be able to update overlap regions correctly (to exchange the boundary data) and to optimize the update procedure. Within the solution process of the user program, the update of the overlap regions of all blocks is then performed by the call of a single CLIC routine. In that call, the user specifies the multigrid level and can choose the number of grid functions to be simultaneously exchanged. Finally, CLIC performs the computation of global values (for example residuals) and writes output, which is generated by node processes to files. An important fact for the development and management of user programs is that a sequential version of CLIC exists. A user program can be sequentially executed with the same interfaces as in the parallel case.

4 Results

4.1 Machines

The MIMD computers for which the test problems have been evaluated are listed below, together with the (standard) compiler flags used.
• IBM PWR2 is an SP2 with thin power2 nodes at GMD. The compiler flags: -O -qarch=pwr2 -qtune=pwr2 are used.
• IBM PWR1 is an SP2 (Software and network) with power1 processors at ANL, USA. Compiler flags were -O -qarch=pwr -qtune=pwr -qhssngl.
• Intel Paragon at KFA, Jülich, Germany was accessed with compiler flag -O.
• NEC Cenju-3 also at GMD with compiler flag -O. This machine is also presented in [3].

The codes also ran on the following (parallel) vector computers:
• Cray C90 a parallel vector machine with up to 12 processors. The test problems are run with auto-tasking, where the compiler does the parallelization, and with CLIC.
• Cray J90 a parallel vector machine with up to 8 processors.
The test problems are likewise run with auto-tasking and CLIC. On the Crays standard compilation options are used.
• NEC SX3 a one processor system at DLR in Göttingen, Germany. This is the production machine on which the sequential code runs (Precompiler used is fopp).

It should be noted that the FLOWer code contains a vector optimization option, whereas no special optimization is carried out on the cache based machines, currently. However, the vector optimization is switched off on the latter platforms, so that it does not influence the performance there. On Intel, Cenju-3, SP2's 32-bit results are obtained, and 64-bit results are obtained on SX3, C90 and J90.

4.2 Euler wing flow problem

A first test example is the computation of a 3D Euler flow around a NACA0012 wing. The Mach number M_∞ and the angle of attack α are chosen as follows $M_\infty = 0.6$; $\alpha = 0°$. This problem is tested with two grids consisting of approximately 40000 and 320000 cells. For both grid sizes the time measured is the wall clock time for 100 multigrid W-cycles with 3 multigrid levels.
The small problem. The first grid considered contains $160 \times 32 \times 8$-grid cells, and is partitioned into 1, 4 and 8 equally-sized blocks (the problems are named gmd1bsmall, gmd4bsmall and gmd8bsmall). Figure 1 shows a part of a block-structured grid around the NACA0012-wing configuration and a partitioning into 8 blocks. The results of this small example obtained with CLIC are presented in Table 1.

Table 1: Wall clock time results obtained with message passing for the small Euler problem.

	FLOWer 1.p.6.1					
	NEC	IBM	IBM	Intel	Cray	
Example	Cenju-3	PWR1	PWR2	Paragon	J90	C90
gmd1bsmall	1962	1081	722	5392	364	84
gmd4bsmall	514	338	224	1444	144	41
gmd8bsmall	302	304	294	865	80	23

The communication and arithmetic overhead is visible for the IBM SP2, when the grid is split into 8 blocks. The wall clock times for the 8 block case are worse than the times for the 4 block case. The times for the NEC Cenju-3 machine are within the range of IBM results. The Intel results are worst for this test case.

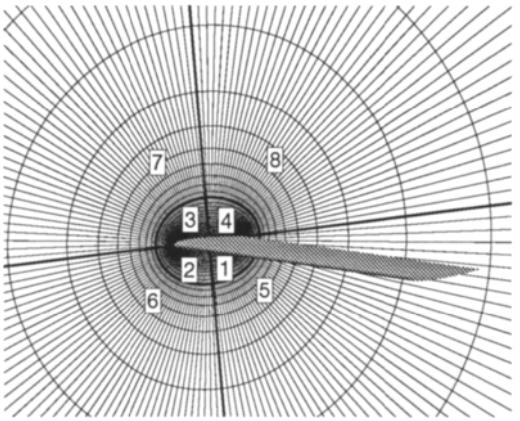


Figure 1: *Grid around the NACA0012-wing configuration and the partitioning into eight blocks. (source: DLR)*

It is now interesting to compare these results with those obtained on the vector machines for gmd1bsmall. The production machine NEC SX3 runs the problem in 21 seconds. On the Cray machine the compiler now takes care of the parallelization. The Cray C90 with one processor is completed in 71 seconds, with 8 processors in 16 seconds and with 12 processors in 14 seconds. The Cray J90 is about 4.5 to 5 times slower than the C90: one processor takes 332 seconds and 8 processors 74 seconds. For this small problem the production machine is hard to beat. Furthermore, the difference between the message passing and vector results is significant. One should keep in mind that the partitioned grid (gmd4bsmall and gmd8bsmall) consists of more grid points than the single block grid, because the interior boundary points are in at least two processors. The Cray results with auto-tasking are about 15% better than the results obtained with CLIC. This difference is due to the fact CLIC is a pure scalar code. Also for a single block there is communication between host and node (transfer of convergence results) and the update on the cut is performed by CLIC.

The medium-sized problem. The second grid considered around the NACA0012 wing contains $320 \times 64 \times 16$-grid cells, and is partitioned into 1, 4, 8, 16 and 32 equally-sized blocks (the problems are called gmd1bmedium, ..., gmd32bmedium). The results with message passing of this larger example are shown in Table 2. Some results are missing for MIMD machines in Tables 2 and 3, because the problem did not fit into the memory of the processors. Due to swapping a comparison is not valid. Furthermore, a result for gmd16bmedium is obtained on a Cray J90 with 16 processors. Here 17 tasks (the node processes and the host) are running on 16 processors.

The anomalous time in the PWR1 curve for 8 blocks is caused by a different cache behavior for different numbers of processors. The IBM SP2 and the Cray machines are the fastest machines for this problem. The NEC machine is also an interesting new competitor in

Table 2: Wall clock times with message passing for the medium-sized Euler problem

	FLOWer 1.p.6.1					
	NEC	IBM	IBM	Intel	Cray	
Example	Cenju-3	PWR1	PWR2	Paragon	J90	C90
gmd1bmedium	—	—	—	—	2243	489
gmd4bmedium	3544	2294	1308	—	667	154
gmd8bmedium	1921	1858	702	5103	374	88
gmd16bmedium	982	690	386	2553	229	-
gmd32bmedium	549	407	233	1395	-	-

the parallel computing area. The vector machines perform as follows on gmd1bmedium: The NEC SX3 runs the 100 multigrid W-cycles in 110 seconds. The Cray C90 uses 414 seconds with 1 processor, 75 seconds with 8 processors and 56 seconds with 12 processors. The Cray J90 solves the problem in 2040 seconds on one processor and in 388 seconds on 8 processors. Also for the medium-sized problem it is hard to beat the vector results. Of course, the differences in costs and in peak performance of these machines are significant. For a 32 processor machine with 64 MB per processor the medium-sized problem can also be considered as relatively small. Larger problems would fit into the memory and will show a better comparison for the MIMD machines. It is expected that large examples with more than 6 million grid points will be solved more efficiently on some MIMD machines. For the small to medium-sized problems evaluated here the conclusion is that the vector machines are still hard to beat.

4.3 The wing-body problem

Finally, wall clock times for an Euler flow around a wing-body configuration are compared. The problem is again a medium-sized problem, with approximately 410.000 grid points. The grid consists of $256 \times 40 \times 40$ grid cells. The so-called C-grid is divided into 1, 4 and 8 blocks (called gmdwb1, gmdwb4 and gmdwb8). The flow parameters considered are: $M_\infty = 0.75$; $\alpha = 0°$. Wall clock time is measured, 35 multigrid W-cycles with 4 multigrid levels are taken as test case. After 35 cycles the lift coeffient has converged. The configuration, the grid and the division into 8 blocks is depicted in Figure 2. The performance on the MIMD machines is presented in Table 3, and looks similar as for the previous medium-sized problem. Cray results are not available with message passing.
The vector results are also known for gmdwb1. The solution process on the NEC SX3 took 50 seconds. On one Cray C90 processor with the auto-tasking option the solution process took 189 seconds, on 8 it took 37 seconds and on 12 processors 29 seconds. On one Cray J90 processor the time measured was 891 seconds and on 8 processors it was 162 seconds. It can be seen that the IBM SP2 results for 8 blocks are less than a factor 2 slower than a single processor CRAY C90 for this problem.

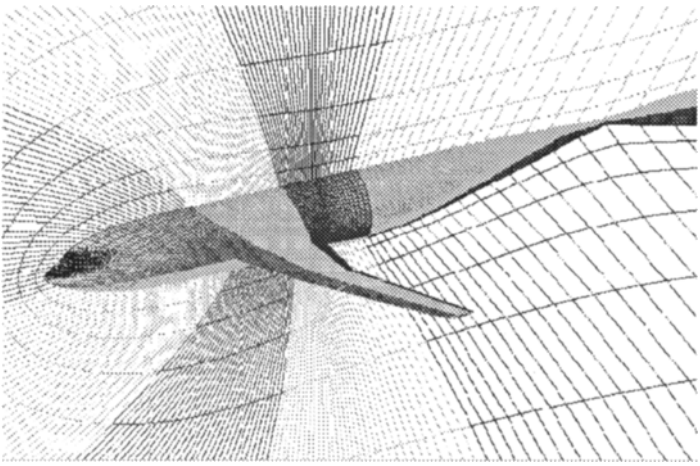


Figure 2: *The domain for the wing body Euler test problem with the division into eight blocks (source DLR).*

Table 3: Wall clock time results obtained with message passing for the medium-sized wing body Euler problem

	FLOWer 1.p.6.1			
Example	NEC/Cenju-3	SP2-PWR1	SP2-PWR2	Paragon
gmdwb1	—	—	—	—
gmdwb4	1508	907	559	—
gmdwb8	819	734	321	2191

5 Conclusions and outlook

In the present paper some results have been presented for providing efficient parallel 3D Navier-Stokes solvers. Existing sequential production codes have been parallelized with a high-level communications library CLIC. For 3D test problems the performance of the CFD code FLOWer on vector and parallel computers has been evaluated. For small and medium-sized problems it is not easy to beat the vector machines, although the MIMD machines already perform very satisfactorily for the problems investigated. The cache optimization of several routines can still be improved. Of course, the peak performances of the machines are totally different as are their costs. It is expected that for larger problems and for more complicated grids the parallel machines will show a better comparison.
In future papers it will be shown that CLIC is able to handle 3D grids with singularities. This means that wing-body-engine-pylon problems, where grid singularities appear, can be solved efficiently on parallel machines. A large wing-body Navier-Stokes test example

with more than 6 million grid points will be solved. Up to now interior boundary points are solved in at least two blocks, which causes an increase in calculation time when a grid is partitioned into many blocks. In the future this could be changed by giving certain processes update rights on interior boundary points. Next year, the library will be extended to adaptive grids (i.e. hierarchies of block structures). This will include routines, that create and manage adaptively refined new grid levels, perform a load-balanced dynamic mapping and perform data re-distribution during adaptive multigrid.

Acknowledgements

The authors thank R. Vogelsang for providing the Cray results, A. Findling and Ch. Lantwin for the NEC SX3 results. Furthermore, the use of the IBM SP machine at the Argonne High-performance Computing Research Facility and the Intel Paragon at the Zentralinstitut für Angewandte Mathematik of the Research Center in Jülich are greatfully acknowledged.

References

[1] R. Calkin, R Hempel, H.C. Hoppe and P. Wypior, Portable Programming with the PARMACS Message-Passing Library. *Parallel Comp.* **20**, 615–632 (1994).

[2] B. Eisfeld, H-M Bleecke, N. Kroll and H. Ritzdorf, *Structured Grid Solvers II: Parallelization of Block Structured Flow Solvers.* In: VKI Series 'Parallel computational Fluid Dynamics', Von Karman Institute, Rhode St. Genèse, Belgium, May (1995).

[3] R. Hempel, R. Calkin, R. Hess, W. Joppich, U. Keller, N. Koike, C.W. Oosterlee, H. Ritzdorf, T. Washio, P. Wypior and W. Ziegler, *Real applications on the new parallel system NEC Cenju-3.* GMD Arbeitspapier 920, GMD Sankt Augustin, Germany, June (1995). Submitted for publication.

[4] N. Kroll, R Radespiel and C-C. Rossow, *Structured Grid Solvers I: Accurate and efficient flow solvers for 3D applications on structured meshes* In: VKI Series 'Parallel computational Fluid Dynamics', Von Karman Institute, Rhode St. Genèse, Belgium, May (1995).

[5] J. Linden, B. Steckel and K. Stüben, Parallel multigrid of the Navier-Stokes equations on general 2D domains. *Parallel Comp.* **7**, 429–439 (1988).

[6] H. Ritzdorf, *-CLIC- The Communications Library for Industrial Codes.* -User's Reference Manual- GMD, Sankt Augustin, Germany (1995).
(⇒ http://www.gmd.de/SCAI/num/clic/clic.html)
• Information on POPINDA available via www:
⇒ http://www.gmd.de/SCAI/num/popinda/popinda.html

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

The Effect of the Grid Aspect Ratio on the Convergence of Parallel CFD Algorithms

Y. F. Hu, J. G. Carter and R. J. Blake

Daresbury Laboratory, CLRC, Warrington WA4 4AD, United Kingdom

In this paper a parallel SIMPLE based incompressible axisymmetric CFD code is considered. It is found that the grid aspect ratio has a strong influence on the convergence rate of the code on several processors relative to that on one processor. This phenomenon is analysed and is attributed to the strong influence of the cell aspect ratios to the structure of the pressure correction equation — a Poisson-like equation. This in turn affects the effectiveness of the parallel preconditioner for the conjugate gradient algorithm used in solving the pressure correction equation. The problem is further confirmed by studying the Poisson equation. A remedy is suggested and demonstrated to improve the convergence of the parallel CFD algorithm.

1. PROBLEMS WITH LARGE ASPECT RATIO

The code considered is one for calculating steady-state incompressible turbulent chemical reacting pipe flow. A cell centred finite volume approach is used to discretise the equation. The coupling between velocity and pressure is dealt with using the SIMPLE procedure ([1]). The discretised momentum equations are solved with line implicit scheme (using TDMA for the resulting tridiagonal systems). The pressure correction equations, being symmetric, are solved using the conjugate gradient algorithm with Incomplete LU (ILU) factorization as the preconditioner. A fixed number of iterations is used, usually 2 for the momentum equation and 10 for the pressure correction equation.

The code was parallelised using the usual domain decomposition strategy. The computational domain is split into a number of subdomains and each of the subdomains is allocated to one processor. A 'halo' region is included in the subdomains and after some calculations on each subdomain, communication takes place to update the data stored in the halo regions. The TDMA is parallelised by each processor sweeping over the subdomain once, then updating the halo data. The conjugate gradient algorithm is easily parallelised except the preconditioner, for which a block diagonal ILU preconditioner is used. The use of block diagonal preconditioner means that each processor calculates the preconditioner based only on the local data it has, thus the operation is fully parallel. However by doing so the parallel conjugate gradient algorithm is no longer mathematically equivalent to the sequential algorithm.

It is found that for some partitionings of the grid, the parallel CFD algorithm takes a lot more iterations to converge, compared to the code running on one processor. This happens frequently when the cell aspect ratios of the grid are large, and in particular, when the domain is partitioned along the radial direction (also called y-split later on) of a long and thin pipe. In such cases the gain in going parallel is eroded by the degradation of the convergence rate.

To give an extreme example, Table 1 shows the results of using the parallel code on a simple pipe flow of 1 m long with a radius of 2.5 cm. The pipe lies parallel to the x-axis. The flow comes in from one end of the pipe and out of the other end, having a density of 1.29 and viscosity of 0.0001. The inlet velocity is 63.8 m/s, and the flow is assumed to be laminar. The size of the mesh is 64×16, that is, the grid is divided into 62 equal sections in x-direction (plus two dummy cells) and 14 in y-direction (plus two dummy cells). The domain is then split in various ways and assigned on to processors. For example, processor configuration 2×1 stands for the partitioning (x-split) illustrated by Figure 1.

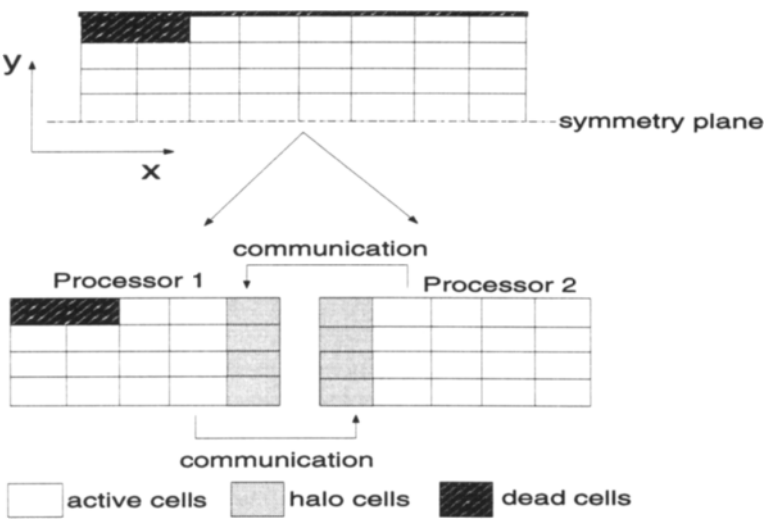


Figure 1

From the table, it is seen that if the domain is split along the axial direction (x-split), the number of iterations taken for the parallel algorithm to converge to a residual of 10^{-6} on several processors is almost the same as the number of iterations taken for the algorithm on one processor. However when the domain is split along the radial direction (y-split), or when box type partitioning is used, the number of iterations taken for the parallel algorithm increases considerably, in most case more than 3 times the number of iterations for the sequential case. This phenomenon is also observed for more

complex problems and for finer meshes, as well as when parallel TDMA is used instead of the parallel conjugate gradient algorithm to solve the pressure correction equations. Although for flow with turbulence, the deterioration in convergence is not as dramatic as that seen in Table 1.

Table 1
Number of iterations taken for the CFD code with different partitionings. Dimension of domain: $1m \times 2.5cm$.

Partitioning	1×1	2×1	4×1	8×1	1×2	2×2	4×2	1×4	2×4	4×4
Iteration Number	368	373	378	386	1200	1134	1162	5704	5614	5637

In the literature, little has been reported about this problem and its cause. Keyes ([2]) reported problems with y-split or box type split, and suggested that it was due to the physics of the particular flow problems studied. Many parallel CFD papers in the literature that worked on long and thin pipe-like geometries with high cell aspect ratios used the x-split only and, in doing so, avoided the problem associated with y-split. However, from the point of view of reducing the communication cost, box-type partitioning is usually more desirable than partitioning along one direction only. Using massively parallel machines, the number of processors may be more than the number of cells along any one direction. Thus, to utilise the parallel computers, a box-type split is unavoidable. Therefore it is very important to investigate the cause of this problem and to come up with some remedies.

2. ANALYSIS OF THE EFFECT OF CELL ASPECT RATIOS

In the SIMPLE procedure, it is necessary to solve a pressure correction equation. It is instructive to analyse the pressure correction equation, since it is a more difficult equation to solve than the momentum equation. The pressure correction equation at a cell P is of the form

$$c_P p'_P = c_E p'_E + c_W p'_W + c_N p'_N + c_S p'_S + d, \tag{1}$$

where c_E, c_W, c_N and c_S are coefficients that correspond to the east, west, north and south faces of the cell respectively, and

$$c_P = c_E + c_W + c_N + c_S.$$

Assuming the mesh is very fine and equi-spaced near the control volume to be considered , it is found that the pressure correction equation (1) is approximately equivalent to

$$\frac{\rho_P}{a_P}\{(\Delta y)^2(2p'_P - p'_E - p'_W) + (\Delta x)^2(2p'_P - p'_N - p'_S)\} = b, \tag{2}$$

with a_P a coefficient of the momentum equation and Δx and Δy the cell size along x and y directions respectively. Clearly (2) has the same form as a discretised Poisson equation.

If the cell aspect ratio $\alpha(=\Delta x/\Delta y)$ is large, then because of the effect of squares in (2),

$$\frac{c_E}{c_P} \approx \frac{c_W}{c_P} \approx \frac{1}{2(1+\alpha^2)} \approx 0,$$

$$\frac{c_N}{c_P} \approx \frac{c_S}{c_P} \approx \frac{\alpha^2}{2(1+\alpha^2)} \approx \frac{1}{2}.$$

As a result, the coupling between north and south is very strong while that between east and west is very weak. Assume the domain is x-split into two as in Figure 1. If a block diagonal preconditioner is used, the coupling between the east and west subdomains, that is, those coefficients c_E and c_W at the processor interface, is essentially ignored. This coupling is very weak anyway from previous analysis, thus its omission will have little adverse effect on the convergence rate. However if the y-split is used, the north-south coupling, which is very strong, is ignored when calculating the diagonally blocked preconditioner. As a result the preconditioner is not a very good preconditioner to the whole system. The usual fixed number of iterations of the conjugate gradient algorithm therefore may not solve the pressure correction equation to a satisfactory accuracy, and the whole CFD code converges slower. This is believed to be the main reason for the deterioration of the performance of parallel algorithms with y-split or box type splits experienced on a long and thin pipe. It is noted that the aspect ratio also effects the momentum equation, however the effect is relatively small compared with the pressure correction equation.

3. ANALYSIS USING POISSON EQUATION

Because the argument applies purely on the interface cells, it implies that if the cell aspect ratio near the interface is not large, the parallel code should not show as significant a deterioration in convergence rate. This is verified by numerical experiment.

Further tests with the use of parallel conjugate gradient algorithms on the solution of Poisson equation, and subsequent eigenvalue analysis, also confirms that it is the large aspect ratios of the interface cells between processors that degrade the performance of the algorithms.

In one such experiment, a Poisson equation on a rectangular domain $[0, XL] \times [0, YL]$ in the (x, y) space is considered. The length of the rectangle along x is fixed to $XL = 1$ and the length along y varies between $YL = 100$ to $YL = 0.001$. The domain is divided into 64×64 cells of equal size. The derivatives on the boundary are assumed to be known. The right hand side of the Poisson equation is set by assuming that the solution is $p(x, y) = x^2 + y^2 + xy$. The number of iterations taken for the residual to become less than 10^{-5} is reported in Table 2, together with the cell aspect ratios. Several partitionings are tested, including splitting the domain into two equal halves with the line $x = XL/2$ (x-split) or with the line $y = YL/2$ (y-split), and box type split with two lines $x = XL/2$, $y = YL/2$. The processor configuration corresponding

to these partitioning are 2×1, 1×2 and 2×2 respectively. As can be seen from the table, with the increase of the cell aspect ratio, y-split takes more and more iterations to converge, x-split generally takes less and less iterations, while the performance of box type split usually follows the worst of that of the former two types of partitionings.

Table 2
The influence of changing domain size to the iteration number of parallel conjugate gradient algorithm on the Poisson equation

XL	YL	Grid size	Aspect ratio	1× 1	1×2	2× 1	2× 2
1	100	64×64	0.01	67	67	226	226
1	10	64×64	0.1	73	73	135	135
1	1	64×64	1	58	86	86	71
1	0.1	64×64	10	63	120	64	120
1	0.01	64×64	100	64	217	64	217

Table 3
The iteration number when the grid is stretched near $y = 0.5$. The aspect ratio shown here is the cell aspect ratio near $y = 0.5$. The grid is uniform elsewhere.

XL	YL	Grid size	Aspect ratio	1× 1	1×2	2× 1	2× 2
1	1	64×64	0.033	64	64	62	65
1	1	64×64	0.1	81	82	92	90
1	1	64×64	1	58	86	86	71
1	1	64×64	10	73	104	86	111
1	1	64×64	30	72	139	86	147

In order to confirm that it is only the aspect ratios of the interface cells that affect the convergence of the parallel codes, in another test the Poisson equation with $XL = 1$, $YL = 1$ and with mesh size of 64×64 is solved, but instead of using a regular mesh, here the grid along x is still equi-spaced, but the grid lines next to the line $y = 0.5$ are shifted, so as to give required aspect ratio of the cells next to the line $y = 0.5$. The rest of the grid are regularly spaced. The results are listed in Table 3, where the aspect ratios shown refer to that of the cells next to the line $y = 0.5$. Clearly with the increase of the cell aspect ratios near the interface, even thought the aspect ratios of the cells that are not next to the interface are close to unity, y-split still takes more and more iterations compared to the sequential case. This confirms that it is the aspect ratios of the cells next to the interface which affect the convergence of the parallel conjugate algorithm relative to the sequential algorithm. The behavior of x-split and that of box type split is also very easy to explain with the same argument.

4. SOME SOLUTIONS

The cause of the performance degradation of the parallel conjugate gradient algorithm on grid with large aspect ratios readily give a possible remedy. Since it is the large

aspect ratios of the cells near the interface of the subdomains that affect the parallel code, it is proposed to coarsen the grid near the interface, thus reducing the cell aspect ratios, in order to improve the convergence.

The procedure is as follows. Denoting by a coarse grid a grid that is coarsened near the interface where the cell aspect ratio is large. Whenever a pressure correction equation is to be solved, the coarse grid equation is first formed and a few iterations of the conjugate gradient algorithm are used to get a good approximation of the solution. This is then interpolated to the original grid and a few CG iterations are applied again to smooth out the interpolation error. The total number of CG iterations will be kept the same as when coarsening is not used.

The coarse grid problem is formed as follows. Since the pressure correction equation behaves like a Poisson equation, for the purpose of calculating the coefficients of the pressure correction equation on the coarsened grid, it is assumed that the equation is of the form

$$c_1(x,y)\frac{\partial^2 p}{\partial x^2} + c_2(x,y)\frac{\partial^2 p}{\partial y^2} = a(x,y).$$

After discretisation, the equation is of the form (1) with

$$c_E = \frac{(c_1)_e \Delta y}{(\delta x)_e}, \tag{3a}$$

$$c_W = \frac{(c_1)_w \Delta y}{(\delta x)_w}, \tag{3b}$$

$$c_N = \frac{(c_2)_n \Delta x}{(\delta y)_n}, \tag{3c}$$

$$c_S = \frac{(c_2)_s \Delta x}{(\delta y)_s} \tag{3d}$$

Now if two cells A (at (i, j)) and B (at (i, j+1)) are lumped into one cell C, then according to (3), assuming

$$(c_1)_e^A(\Delta y)^A + (c_1)_e^B(\Delta y)^B = (c_1)_e^C(\Delta y)^C$$

and

$$(c_1)_w^A(\Delta y)^A + (c_1)_w^B(\Delta y)^B = (c_1)_w^C(\Delta y)^C,$$

the coefficients of the discretised equation on the coarse cell can be calculated using that on the fine cells. That is,

$$(c_E)^C = (c_E)^A + (c_E)^B,$$

$$(c_W)^C = (c_W)^A + (c_W)^B,$$

$$(c_N)^C = \frac{(c_N)^B(\delta y)_n^B}{(\delta y)_n^C},$$

$$(c_S)^C = \frac{(c_S)^A(\delta y)_s^A}{(\delta y)_s^C}.$$

If more than one cells are lumped to form a cell, the coefficients of the coarse cell can be calculated in a similar way.

The source terms on a coarse cell are calculated by lumping the source terms on the fine cells which form the coarse cell.

The idea is implemented and is found to improve the convergence of the parallel algorithms for the Poisson equations as well as the convergence of the parallel CFD code for pipe flow.

Table 4
Iteration number and CPU time of the CFD code with various processor configurations and with or without coarsening on problem 2

Processor configuration	Iterations	Total cpu time	Communication time
1×1	5740	3243	0
1× 2	5840	2067	309
1×2 coarsened	5420	1905	267
2×1	5680	1794	182
2×2	5840	1253	335
2×2 coarsened	5300	1167	320
1×4	5700	1547	519
1×4 coarsened	5480	1483	520
4×2	5820	988	404
4×2 coarsened	5240	920	382
2×4	13300	2448	1065
2×4 coarsened	5480	1026	457
8×1	5700	933	358
4×4	5720	899	470
4×4 coarsened	5460	871	445
8×2	6100	854	419
8×2 coarsened	5320	765	381
16×1	5560	804	396

To give an example, turbulent flow in a sudden expansion pipe is considered. The pipe is 1.0 m long and has a radius of 0.1 m. The inlet volecity is 64 m/s, density is 1.29 and viscosity 0.0001. A 64×32 equi-spaced grid is used. The results are listed in Table 4.

As can be seen from the table, partitioning along the y direction tends to increase the number of iterations needed for the algorithm to converge, although the adverse effect is less dramatic as in the case of the laminar flow problem. Coarsening produces some of the best iteration numbers and least CPU time.

Coarsening however has its limitation. If the cell aspect ratio is extremely large, then lumping a few cells together will not reduce the aspect ratio by very much. How

to implement the coarsening procedure on irregular grid in 3D also needs further investigation.

5. CONCLUSIONS

The problem associated with high aspect ratios when working on parallel incompressible CFD algorithms are analysed. The degradation in convergence comes only when the domain is decomposed along a direction where the cell aspect ratios are high, and is attributed to the strong coupling in the pressure correction equations between subdomains.

Coarsening as a remedy to the problem has been shown to be useful.

Not reported here, the use of block correction ([3-5]) was also tested and found to improve the convergence slightly. Overlapped block diagonal preconditioner was suggested ([6]) to improve the pure block diagonal preconditioner, this and similar ideas were also tried on the current problems but were found to make little improvement.

Currently the coarsening strategy is being compared with other techniques such as the use of Schur complement type preconditioners ([7-8]).

REFERENCES

1. S. V. Patankar, Numerical Heat Transfer and Fluid Flow, McGraw-Hill, 1980.
2. D. E. Keyes, Domain decomposition methods for the parallel computation of reacting flows, Computer Physics Communication, 53 (1989), 181-200.
3. A. Settari and K. Aziz, A generalization of the additive correction methods for the iterative solution of matrix equations, SIAM Journal of Numerical Analysis, 10 (1973), 506-521.
4. S. V. Patankar, A calculation procedure for two-dimensional elliptic situations, Numerical Heat Transfer, 4 (1981), 409-425.
5. B. R. Hutchinson, P. F. Galpin and G. D. Raithby, Application of additive correction multigrid to the coupled fluid flow equations, Numerical Heat Transfer, 13 (1988), 133-147.
6. G. Radicati and Y. Robert, Parallel conjugate gradient-like algorithms for solving sparse nonsymmetric linear systems on a vector multiprocessor, Parallel Computing, 11 (1989), 223-239.
7. T. F. Chan and T. P. Mathew, The interface probing technique in domain decomposition, SIAM Journal of Matrix Analysis and Applications, 13 (1992), 212-238.
8. J. H. Bramble, J. E. Pasciak and A. H. Schaz, The construction of preconditioners for elliptic problems by substructuring, Mathematics of Computation, 47 (1986), 103-134.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Master-Slave Performance of Unstructured Flow Solvers on the CRAY T3D

Roland RICHTER * and Pénélope LEYLAND **
* *CRAY Research (Switzerland) S.A., Scientific Parc (PSE)*
** *Institut de Machines Hydrauliques et de Mécanique des Fluides*
Ecole Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland

Abstract

Performance issues of unstructured mesh solvers with mesh adaptation, for compressible flows, using different programming models and communication libraries are discussed. The partitioning of the grid into subdomains is performed on a master (CRAY YMP), as well as the subsequent adaptations, the solution process is executed on the slave (CRAY T3D), with up to 128 processors. Three flow solvers of different degrees of complexity are used to illustrate the trade-off between synchronisation and redundant calculation necessary at subdomain interfaces. Finally, amongst the test cases presented, a transient regime effect in optimal design is calculated, which requires a high number of runs of complete convergence, illustrating the practical utility of parallel CFD, where CPU time becomes **elapsed** time and allows these series of runs to be made in a very short period of real time.

1 Introduction

Most present day parallel implementation of CFD codes on MIMD type parallel architectures use a master-slave paradigm. The master process is responsible for reading the initial grid, partitioning the problem, communicating with the slaves and writing the final solution. The master can run either on an external *remote front end* machine, or on one of the processing elements of the MIMD array itself, which could be dedicated to the spawning of data.

One of the great advantages of using unstructured meshes for CFD applications is the possibilities of solution adaptation of the meshes using local grid refinement/derefinement. This enables a highly precise solution on a minimal number of nodes and allows the mesh to "follow" the physical phenomena, which is particularly important for unsteady simulation. Such techniques are also an important "optimisation" strategy for improving initial grids of their different geometrical and physical irregularities [6, 7]. This dynamic mesh adaptation has proved to be extremely efficient in both academic and industrial applications [4, 6], however it remains a challenging task to perform this completely within a parallel environment.

In its simplest form, mesh refinement takes place on the master; the new grid is then completely repartitioned and sent to the slaves to continue the calculation. Such an approach remains efficient if the number of grid refinements is small compared to the number of iterations required to solve the fluid problem. It also considerably simplifies any possible coupling with the CAD surface mesh generation package. However in order to use efficiently and to progress in new parallel technology, it is necessary to advance this situation by using the master only for data management; the grid adaptation, the partitioning and the complete computational cycle should take place on the parallel architecture itself.

2 Parallelisation of CFD Solvers on Unstructured grids

Parallelisation of CFD Solvers for a distributed memory environment, such as on the CRAY T3D, requires first a domain decomposition of the problem by some appropriate partitioning algorithm. The computational domain is thus partitioned into a certain number of submeshes, which can be larger than the number of processors. Common nodes to adjacent submeshes are repeated within each submesh so that every submesh keeps a consistent and independent structure. The parallelisation itself is done on a subroutine level with dynamic memory allocation to facilitate programming, but the algorithm remains *global*. The choice of partitioning technique has been kept, in our case, to the simplest one : i.e. *Recursive Coordinate Bisection* (RCB). This method is purely based on the coordinates of the mesh points where the bisection is performed parallel to the $x = 0$, $y = 0$ (and $z = 0$) planes, depending of eigenvector of the moment matrix of the mesh. An example of such partitioning is shown in Figure 4, which illustrates a partial view of an adapted mesh for a transonic flow over a NACA0012 profile. Other standard methods are *Recursive Geometric Bisection* (RGB) or *Recursive Spectral Bisection* (RSB) which includes some of the mesh connectivity information. Communication is finally achieved by exchanging data between inter-domain nodes.

2.1 Domain Decomposition Issues

Domain decomposition is performed usually *a priori* on the Master. The initial grid, (single or multiblock), is partitioned into a certain number of subdomains, to redistribute the information load as uniformly as possible throughout the processing elements. If necessary several subdomains can be allocated to a same processor, (this is particularly useful for structured grids [5]). Each subdomain acts then independently, with exchange of information at the interfaces between the domains in order to update the solution process [1, 5]. The way in which this communication phase is implemented depends partly upon the type of numerical scheme involved to solve the governing equations, and upon the precision required. The exact balance between the type of "interface overlay information" to be exchanged within the *update process* of the individual subdomain solutions and between the relative cost of those synchronisation points can vary considerably, as will be detailed below.

2.2 Numerical Schemes on Unstructured Meshes

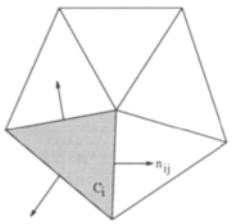
In this paper we consider the simulation of compressible flow phenomena, where the governing equations can be written in conservation form as :

$$\frac{\partial \mathcal{F}(W)}{\partial t} + \nabla \cdot \mathcal{F}(W) = 0$$

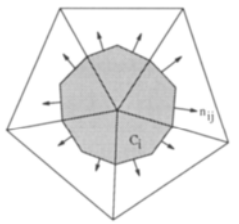
The natural discretisation of such systems of conservation laws requires the evaluations of the flux balance integrals within typical control volumes. The data structure underlining the spatial approximation within numerical schemes on unstructured meshes can be divided roughly into three categories :

- Schemes where the calculation of the variables at the simplicial nodes require knowledge of all neighbouring nodes values using thus *direct control volumes*; most weighted residual cell vertex schemes (mixed finite element / finite volume methods) fall into this category.
- Schemes for which the variables are calculated via numerical fluxes which are an approximation of the flux integral in the normal direction n_{ij} across the interface between computational cells, the consistent *dual control volumes*; these schemes are mainly finite volume methods.

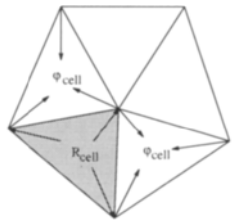
- Schemes for which all information for the calculation of the solution is derived within each element, this is the case of the so-called *compact approximation* schemes; for instance most truly finite element methods.



Direct control volume.



Dual control volume.



Compact approximation.

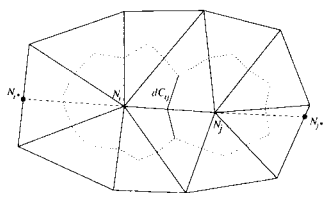
The scheme is then defined via a **numerical flux function** $\Phi_{F_{ij}}$ which evaluates the flux integral in the normal direction n_{ij} across interface between cells (*direct or dual control volumes* $\mathcal{C}_i$) :

$$\int_{\partial \mathcal{C}_{ij}} F(W) \cdot n_{ij} \, d\sigma \; = \; \Phi_{F_{ij}}(W_i, W_j)$$

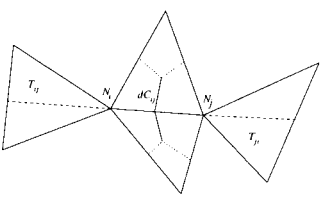
It results that any scheme can be written as the sum of an average plus a numerical dissipation, which can be of artificial or numerical type :

$$\Phi_{F_{ij}}(W_i, W_j) \; = \; \frac{1}{2}[F(W_i) + F(W_j)] \; + \; \Phi^*_{F_{ij}}(W_i, W_j)$$

For hyperbolic systems of conservation laws, standard finite element methods are unstable and need to be stabilised via upwinded test functions or via artificial viscosity formulations. For the equivalent finite volume methods, either upwind solvers are adapted in the direction n_{ij}, or consistent artificial viscosity terms are added. The sufficient precision of the scheme is obtained by gathering information via virtual nodes $N_{i^\star}$ and $N_{j^\star}$, as for the construction of second and fourth order dissipation terms for centred schemes, or via up and down wind elements for a higher order reconstruction in the case of approximate Riemann solvers or upwind finite volume schemes :



Virtual points $N_{i^\star}$ and $N_{j^\star}$.



Up and down wind elements.

In the present paper a Lax-Wendroff scheme based on *direct* control volumes, and two schemes using the *dual* control volumes formulation : a Jameson type centered scheme and an Osher scheme [6, 8], will be compared.

2.3 Domain Decomposition Interfaces

There exist mainly two distinct possibilities to define the domain decomposition interface, which are valid for structured and unstructured meshes. For the first one, the partition interface between the subdomains follows the grid lines, so that each element (in the present case : the triangles) belongs to a unique partition, only interface nodes have to

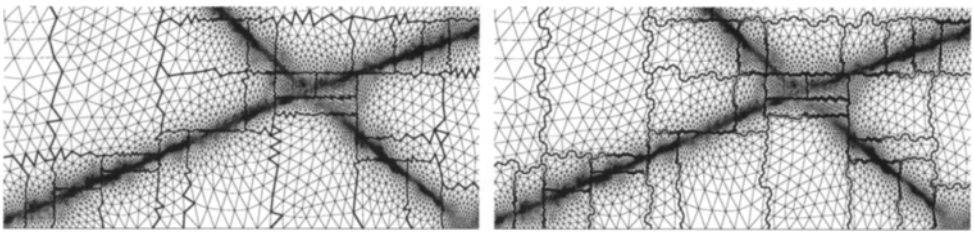


Figure 1: *Zoom over domain decompositions using both interface definitions; along the grid lines* (left) *and between the nodes* (right).

be duplicated in the adjacent domains to obtain the consistent submeshes. The second possibility cuts the mesh between the nodes producing thus an element layer which is duplicated, in this case every node belongs to a single partition (see Figure 1).

Commonly these two domain decomposition interfaces are known as *non-overlapping* when the partitioning is performed along the grid lines and *overlapping* in the other case. Such names can be confusing, especially in the case when additional ghost cell lines are required for the precision of the scheme or due to the type of control volume chosen. In this paper, the terminology "direct decomposition" and "dual decomposition" will be used, referring to the direct and dual control volume interface boundaries. Moreover, in the case of the necessity of additional ghost cell lines, the *direct* decomposition will produce always **even** numbers of overlapping layers and the *dual* decomposition only **odd** numbers.

This mean that schemes based on *direct* control volumes, such as the Lax-Wendroff one, can evaluate the physics without overlapping layers using a *direct* decomposition, or with a single layer in the *dual* case. Schemes using *dual* control volumes with second order space accuracy require two or three overlapping layers at these internal boundaries. The two decompositions are illustrated in Figure 2.

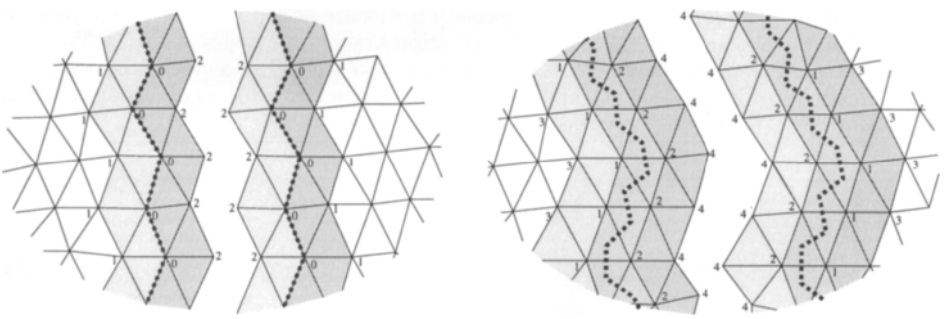


Figure 2: *Direct* (left) *and Dual* (right) *decomposition with one layer of ghost cells.*

The complete update process for schemes like the Lax-Wendroff one using *direct* decomposition requires thus only flux and time step contributions to be exchanged at nodes "0" (Figure 2-*left*), imposing two synchronisation points per time iteration. (This technique is exhaustively described in [1, 2].) For the other case, when *dual decomposition* is used, only the physical values themselves have to be transferred from "1" to "2", (Figure 2-*right*) reducing thus the number of synchronisation points to one per iteration. However, the drawback of this second implementation is, that the flux contributions within the overlapped elements need to be calculated redundantly on several processors.

For schemes using *dual* control volumes the update process is quite similar. In the first case flux contributions are exchanged at "0" and the physical values transferred form "1"

to "2" (Figure 2-*left*), producing again 2 synchronisation points. In the second case, an additional "3" to "4" transfer is performed, which can be done simultaneously transferring "1" to "2" (Figure 2-*right*), resulting in once again only one synchronisation point.

In both cases, changing from *direct* to *dual* decomposition produces a reduction of the number of synchronisation points with an increase of the redundant calculation. As it will be shown in the next section, the choice between the strategy adopted gives varying degrees of performance improvement depending on the complexity of the scheme, the speed of the communication library and the hardware used.

2.4 Performance Comparisons on the CRAY T3D

The comparison of the communication cost for both decomposition strategies is given in Figure 3, using the three different schemes: a Jameson type centred scheme on dual control volumes with second and fourth order artificial viscosity, a compact cell weighted residual Lax-Wendroff scheme on direct control volumes and a complex approximate Riemann solver using Osher's scheme on dual control volumes. The schemes are ordered here according their computational complexity and floating point operation per time iteration.

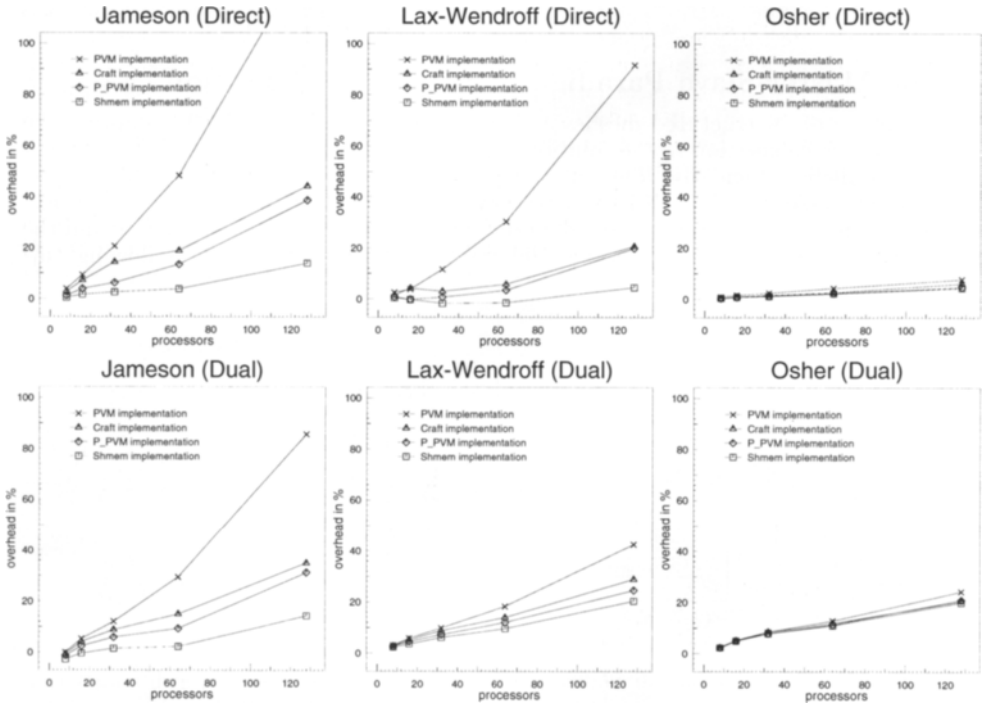


Figure 3: *Communication libraries comparisons using different decompositions on CRAY T3D.*

The test problem used correspond to a simple channel flow, presenting 25 168 nodes and 49 932 elements, on which 1000 iterations are performed per run. The performance analysis is expressed in terms of overheads, defined as :

$$overhead = \frac{Npes * Time_{Npes} - Time_{1pe}}{Time_{1pe}}$$

The basic structure of the code is kept unchanged, but the transfer of data between submeshes is undertaken via the different message-passing libraries: the PVM library,

based on the Oak Ridge National Laboratory version 3, using `pvm_send`, (denoted in Figure 3, as `pvm`), its optimised version for short messages `pvm_psend` (`p_pvm`), CRAY's implicit data parallel model (`craft`) and CRAY's shared memory access library (`shmem`).

The `shmem` library provides the best performance in all cases, but puts also more weight on the programmer's side, especially when PE to PE synchronisation is necessary. `Craft` and `p_pvm` are also good alternatives, the first for its writing simplicity and the other for it portability.

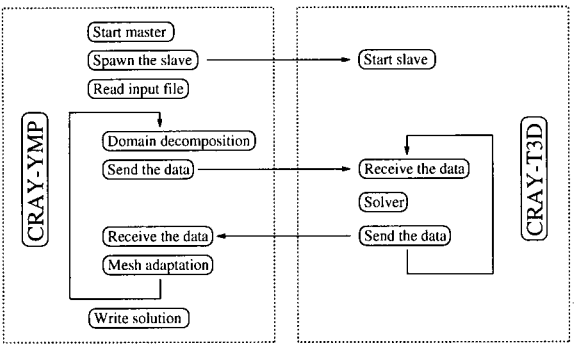
It results that, the simpler the scheme becomes, the more a fast communication library is needed. For the more complex schemes, the parallel behaviour can be improved by trading the expensive redundant calculation by further communication with additional synchronisation points.

Direct decomposition with fast communication (`shmem` or `p_pvm`) seems to be the best solution, for the cases studied here. However, this result should also be valid for other schemes such as, for example, cell centred ones on structured grids, where a *direct* decomposition would mean; cutting the domain along the physical values location.

To perform this comparison accurately, the code has been optimised for CRAY T3D single PE performance, with cache alignment issues which are crucial for the small cache of the used RISC processor, as also its instruction cache sensitivity [9].

3 The Master-Slave Paradigm with Mesh Adaptation

The advantage of unstructured meshes are their flexibility of dynamically adapting the mesh during the calculation to the solution process, however such procedures are complicated to parallelise efficiently. The auto-adaptive mesh algorithm is thus implemented, in the present case, within a Master-Slave environment for distributed execution. The domain decomposition and the mesh adaptation are performed on the *Front End* and the solution process is executed on the parallels processors, as shown in the following flow chart :



As an example of this Master-Slave paradigm, a complete run with mesh adaptation and solution cycle for a lifting airfoil, at transonic regime of $M_\infty = 0.80$ and $\alpha = 1.25°$ is given in Figures 4. The initial mesh is extremely coarse (1704 nodes and 3332 elements), but within 5 passes of adaptation using refinement/derefinement, and geometric/physical optimisation [6, 7], a highly optimised grid (8136 nodes, 16078 elements) has been obtained, which allows a precise capturing of the weak windward side shock.

The timings for this complete run are given, in Figure 5, according to the number of processors used on the CRAY T3D. It can be seen that the total elapsed time equivalent here to the `total T3D time` decrease well with the number of parallel processors. On the *Front End* side, the grid partitioning and mesh adaptation phases remain quite constant, as well as the data transfer time between the master and the slaves. However the part

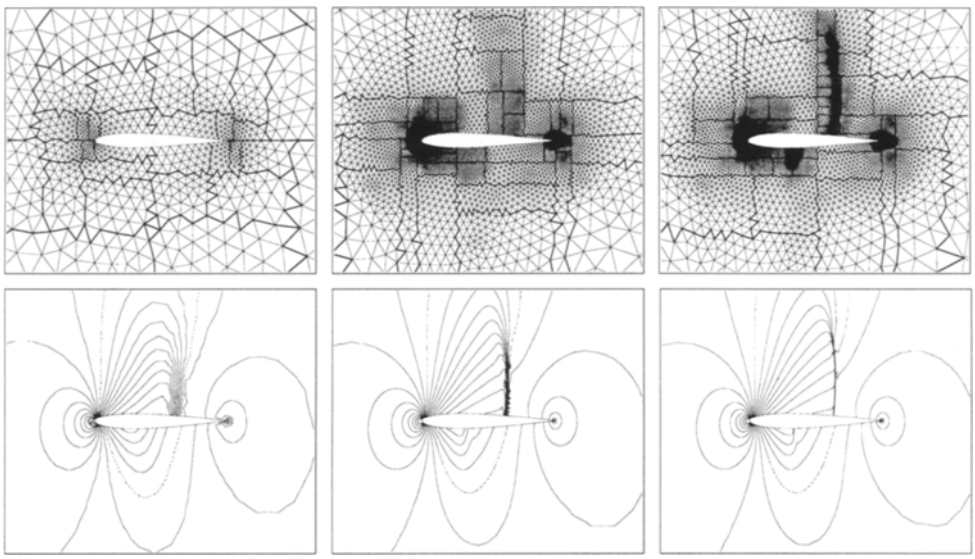


Figure 4: *Evolution of the C_p iso-lines around a NACA0012 at $M_\infty = 0.80, \alpha = 1.25°$, initial mesh (1704 nodes/3332 elements), final mesh after 5 refinements (8136 nodes/16078 elements).*

which increases with the number of processors is the preparation of the data, which means the renumbering of the nodes and elements, so that each processor obtains a consistent submesh with its own local numbering and the required information for the later inter-domain exchanges. The overhead resulting out of the whole process becomes thus a linear function of the number of processor. In the present case, the efficiency drops already to 70% on 64 processors.

Another test case over the same profile, but in supersonic regime at $M_\infty = 0.95$ and $\alpha = 0.0°$, has also been performed. This regime has the particularity of a fish tail shock structure within the wake and an emanating normal shock. The correct position of this shock necessitates a very high density of mesh points not only within the shocks, but also around the profile itself in order to capture the successive expansion fans [7]. Starting again from the same coarse mesh, 5 passes of adaptation were necessary until the transient effects stabilised out and the shock positions were correct (Figures 6). The final mesh becames thus extremely fine with 54 469 nodes and 108 621 elements.

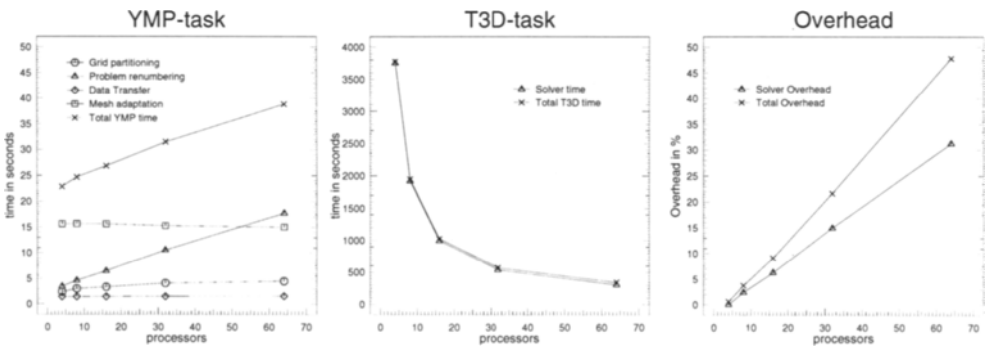


Figure 5: *Timings for the complete NACA0012 at $M_\infty = 0.80, \alpha = 1.25°$ run.*

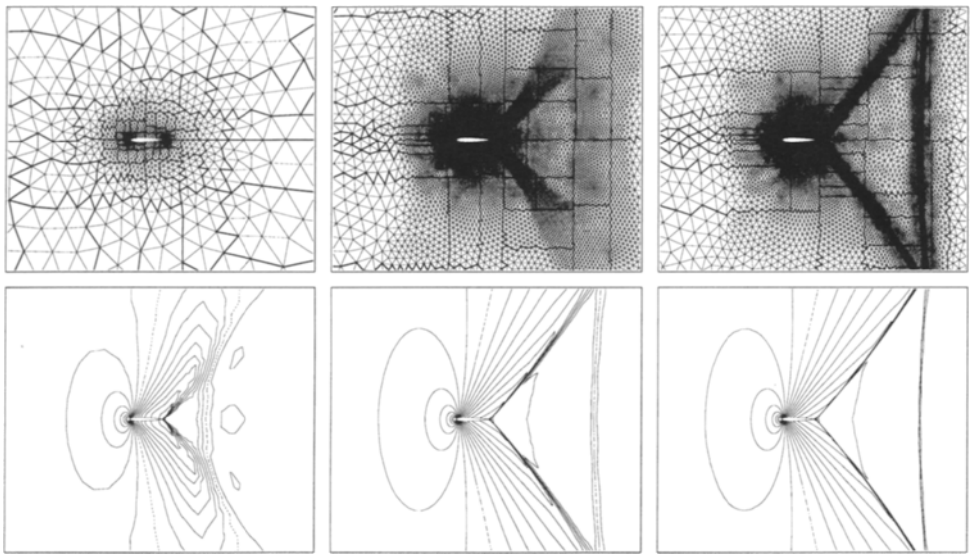


Figure 6: *Evolution of the Mach iso-lines around a NACA0012 at* $M_\infty = 0.95, \alpha = 0.0°$, *initial mesh (1704 nodes/3332 elements), after 5 refinements (54469 nodes/108621 elements).*

The timings (Figures 7) looks very similar to the previous ones, even if the size of the problem has considerably increased. The main difference is that the larger mesh points number per subdomain, improved well the efficiency of the solver part, but the global overhead remains the same linear function with the number of processors.

On a third test case, solving a flow over a multi element airfoil [7], where also 5 adaptation phases are needed for an accurate solution, the final mesh size became around 20 000 nodes and 40 000 elements, however the timings showed again the same behaviours. The total overhead increased by 0.65 % per processor.

It results that, the total overhead for such a Master-Slave environment, with grid partitioning and mesh adaptation performed on the *Front End*, is a linear function of the number of parallel processors, but **independent** of the number of mesh points. In order to go to truly parallel execution, and prepare future work on unsteady automatic dynamical mesh adaptation in 3D, the above Master-Slave paradigm has to be improved. The Master should be reduced to a master region, responsible only for initial input and final output,

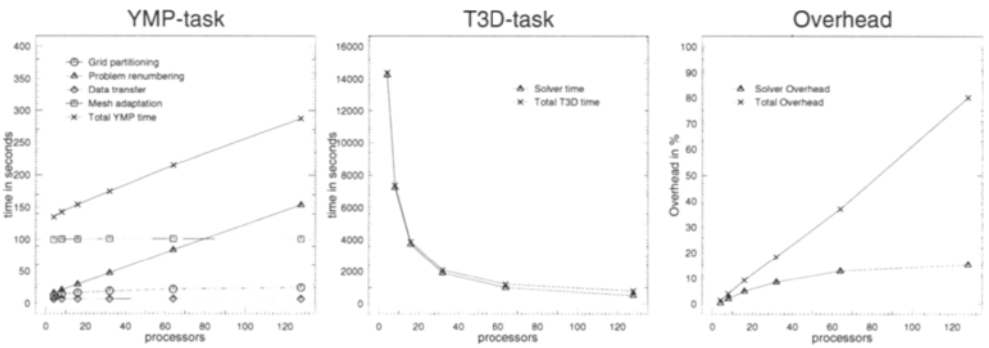


Figure 7: *Timings for the complete NACA0012 at* $M_\infty = 0.95, \alpha = 0.0°$ *run.*

the mesh adaptations need to be performed in parallel and the domain decomposition should be replaced by a redistribution scheme to enable dynamic load balancing.

4 Transient Design - a Test Case for Parallel Execution

During an optimum design procedure on airfoils, the perturbation of the surfaces can generate distinct solution branches showing an hysteresis effect in the lift/incidence polar, when varying the angle of attack back and forth [3, 7]. The grid must be extremely fine and regular around some parts of the airfoil, to be able to obtain these effects. It is thus a challenging test case for mesh adaptation procedures. The freestream Mach number is fixed at 0.78 and an initial series of computations is performed up to complete convergence (up to 30000 time step iterations) for increasing angle of attack from $-0.2°$ to $-0.7°$. The different lift coefficients are recorded and shown in Figure 9. Then for the solutions around a C_L of 0.6 and angle of attack around $-.43°$, a second series is initialised by taking the solution of the previous slightly inferior angle of attack. As the mesh adaptation is dynamically linked to the solver, the essential characteristics of the changing solution are taken into account and each calculation is thus made on its own specific adapted mesh (Figure 8). In all, up to 30 calculations are necessary to perform accurately the complete lift polar. This procedure was executed on a CRAY T3D using 128 processors, each run taking approximately 800 seconds, the whole series was thus possible to be executed on one single afternoon. This is a demonstration of the practical utility of parallel CFD, where CPU time is now **elapsed** time; often these codes are less efficient than their serial origins in terms of single PE megaflops, but the engineer can produce a significant quantity more calculations in the same period of office time.

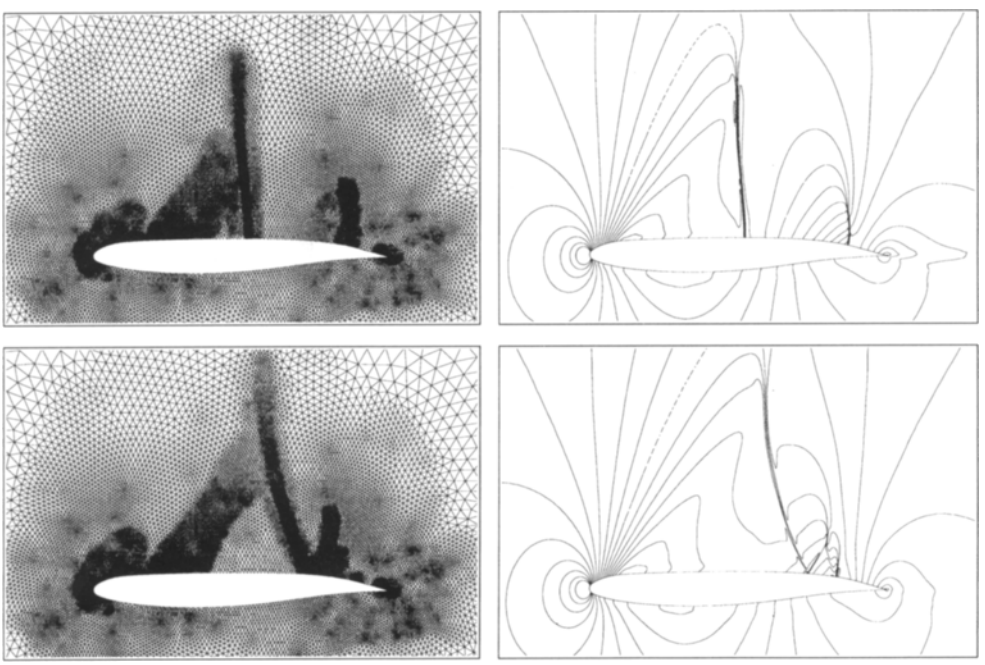


Figure 8: *Different Meshes (with around 22 000 nodes and 40 000 elements) and Mach values corresponding to hysterisis effect of the lift polar, for the J-78 airfoil at* $M_\infty = 0.78$, $\alpha = -.47°$

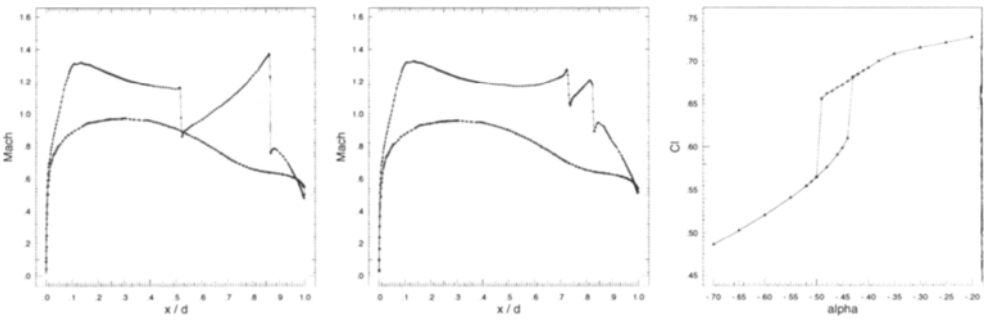


Figure 9: *Body Mach values for the two distinct solution branches of the hysterisis effect of the lift polar, for the J-78 airfoil at* $M_\infty = 0.78$, $\alpha = -.47°$

5 Conclusions

The present work has shown that a Master-Slave environment for solving steady state flows with auto-adaptive mesh adaptation is feasible as long as the number of refinements is low. In 2D, there seems to be an upper limit of 128 PE's, if the efficiency of the Master-Slave implementation using a CRAY YMP and a CRAY T3D should be at least 60 %. The overhead is manly due to the renumbering of the nodes/elements and the setup of the communication interfaces. It increases linearly with the number of PE's at 0.6 to 0.7 % per PE, but is independent of the number of mesh points used. Domain decomposition and/or mesh redistribution should thus be done on the parallel machine, even when no mesh adaptation is required.

References

[1] L. Bomholt and P. Leyland; *Implementation of unstructured finite element codes on different parallel computers.* Proceeding Parallel CFD'93, 1993.

[2] C. Farhat and S. Lanteri; *Simulation of compressible flows on a variety of MPP's.* INRIA Technical Report, RR-2154, 1994.

[3] A. Jameson; *Airfoils Admitting Non-unique Solutions of the Euler equations.* AIAA 91-1625, 1991.

[4] P. Leyland, et al.; *Dynamical mesh adaptation criteria for accurate capturing of stiff phenomena in combustion.* Int. Journ. Num. Meth. in Heat and Mass Transfer, 1993.

[5] P. Leyland, J.B. Vos, V. Van Kemenade and A. Ytterstrom; *NSMB: A Modular Navier Stokes Multiblock Code for CFD.* AIAA 95-0568, 1995.

[6] R. Richter; *Schémas de capture de discontinuités en maillage non-structuré avec adaptation dynamique.* PhD Thesis, EPFL, 1993.

[7] R. Richter and P.Leyland; *Distributed CFD using Auto-Adaptive Finite Elements* ICASE/LaRC Workshop on Adaptive Grid Methods, Hampton Virginia, 1994.

[8] R. Richter and P. Leyland; *Entropy Correcting Schemes and Non-Hierarchical Auto-Adaptive Finite Element Type Meshes* Int. Journ. for Num. Meth. in Fluids, 1995.

[9] R. Richter; *Instruction Cache Sensitivity on the CRAY T3D.* Technical report, 1995.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Runtime Volume Visualization for Parallel CFD

Kwan-Liu Ma*

Institute for Computer Applications in Science and Engineering
Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23681-0001

1. INTRODUCTION

For many scientific and engineering problems, computational fluid dynamics (CFD) has become increasingly popular as a means to gather information for design and analysis purposes. Massively parallel computing offers both the computing power and memory space required to attain the desirable accuracy and turnaround time for CFD calculations. Traditionally, monitoring of typical parallel CFD calculations is done by transmitting, at regular times, a subset of the solutions back to a host computer. If the user just wants to track a few numbers at some particular spatial positions, then only a small amount of data is transferred. However, if the user needs to analyze or *visualize* the overall flowfield, the amount of data transferred could be enormous, possibly ranging from several megabytes to gigabytes. Acquiring the whole distributed domain of data involves expensive operations. Storing and postprocessing the data on another computer could also be problematic.

A better approach is to analyze data in place on each processor at the time of the simulation. The locally analyzed results, which is usually in a more economical form, e.g. a two-dimensional image, are then combined and sent back to a host computer for viewing. Scientific visualization is an effective data analysis method making use of computer graphics techniques, and volume rendering has been recognized as one of the most direct ways for visualizing three-dimensional data. This paper describes the design of a parallel volume rendering (PVR) library which renders in-place distributed data on a rectilinear grid. There are special considerations for such design and implementation. Due to limited space, our discussion focuses on the PVR algorithm. In [1], a thorough discussion is given for the library design of a parallel *polygon* renderer.

We performed tests for runtime visualization of a three-dimensional Navier-Stokes solver, on the Intel Paragon XP/S using from 8 to 216 processors. The interactive visual response achieved is found to be highly desirable. The realistic pictures of the overall flowfield help not only monitor but also understand the simulation. Performance studies show that the parallel rendering process is scalable with the size of the simulation as well as with the parallel computer. Although our current implementation only handles data on a rectilinear grid, the design principles of the library can be generalized to handle unstructured or curvilinear grids as well. A PVR algorithm developed for unstructured-grid data is described another paper [2].

*Financial support is provided by NASA Contract NAS1-19480.

2. VOLUME VISUALIZATION

Direct volume rendering is a powerful visualization technique. It can render the flow-field realistically as a semi-transparent gel or smoke-like cloud. However, direct volume rendering involves very computationally intensive calculations. Interactive rendering of large data set can be achieved by using a massively parallel computer.

There are two approaches for direct volume rendering: projection and ray-casting. For this work, ray-casting is used because it is easier to parallelize and implement; furthermore, it can also render cut-planes and iso-surfaces. In the ray-casting approach, an image is constructed in image order by casting rays from the eye, through the image plane and into the data volume. One ray per pixel is generally sufficient, provided that the image sample density is higher than the volume data resolution. The data volume is sampled along the ray, usually at a rate of one to two samples per voxel (volume element). At each sample, a color and opacity is computed by interpolating from the data values. For parallelepiped element, there are eight vertices and trilinear interpolation is often used. The final image value corresponding to each ray is formed by compositing, front-to-back, the color as well as the opacity of the samples along the ray. The color/opacity compositing operation is *associative*, which allows us to break a ray up into segments, process the sampling and compositing of each segment independently, and combine the results from each segment via a final merging step. This is the basis for our PVR algorithm [3].

3. DESIGN CONSIDERATIONS FOR A PVR LIBRARY

The design and implementation of a software library involves several key issues such as the application programmer user interface (API), portability, performance and versatility. The library should have a simple interface that does not intimidate the user. A good library should mask differences across system software and hardware platforms through abstraction. Good performance is essential so the user will not be tempted to write custom code. Ideally, a library should improve performance beyond what an average user could easily achieve without it. Finally the library should adapt to unforeseen situations, such as low memory restrictions, and to more sophisticated application problems. Based on the above principles, design considerations essential to implementing a PVR library include:

- Parallelizing direct volume rendering:
  - Data distribution
  - Load balancings
  - Memory constraints
  - Parallel resampling
  - Parallel compositing
  - Scalability
- Portability
- Image output and display
- Library interface

Like most other parallel applications, parallelizing direct volume rendering is a divide and conquer process. The goal is to distribute both data and computation among available processors. There are basically two approaches for parallelizing direct volume rendering: one is to separate the ray-casting (i.e. resampling) process from the compositing process [3]; the other is to overlap them [8,2]. The first approach has been selected for rendering regular data because it is straightforward to implement and can render as efficient as the overlapping approach in a runtime visualization setting.

Data distribution is a problem for both the application (parallel CFD) and the visualization (PVR). The design principle is to allow the application programmer to focus on the application program instead of the rendering program. Ideally, a renderer should

be able to process local data regardless how decomposition was done. For both parallel CFD and PVR, it has been shown the best scalability can be achieved by decomposing the domain in such a way as to have sizes in all dimensions as close as possible [9,6]. For example, three-dimensional decomposing (into blocks) is better than one-dimensional (into slabs); for parallel CFD, block decomposition could reduce communication costs, and for PVR, rendering would become less view-dependent. Currently, the renderer can handle one-, two-, and three-dimensional decomposed data, but each data subset must contain voxels that are spatially consecutive; this restriction can be released for irregular data partitioning [2]

Load balancing is an important issue for achieving the maximum parallel efficiency. According to our test results, imbalanced load generally can degrade the overall performance by 20-40%. Dynamic load balancing, if implemented efficiently, should be able to remove at least 50% of the degradation. A few load balancing strategies for PVR have been proposed which require preprocessing of the data, specialized parallel architectures or data distribution schemes [5,8,7]. For time-varying data, preprocessing is generally impractical since the distribution of the interested part of the data can not be decided statically. For an in-place rendering library design, implementing efficient dynamic load balancing strategy is challenging.

The rendering process is separated into local resampling and global compositing. Consequently, no communication is required during the resampling process. The parallel resampling algorithm is based on the author's previous work [3] which requires replicating voxels at boundaries. As most CFD codes use a finite-difference formulation, replication is required anyway for a parallel implementation. Nevertheless, the replication requirement could be removed if resampling were done differently.

Another important issue to consider is the memory constraints of a processor as most applications are memory-intensive. It is usually not the case that the simulation and the renderer can share the data. Additional memory space is needed to store the data to be rendered, rendering parameters and partial images. Normally, the quantities to be visualized are calculated and stored separately by the application, and passed to the renderer. Therefore, a PVR library should have moderate and predictable memory requirements so the application programmer can plan accordingly. The amount of memory space for the renderer is dominated by $[(n_{voxel}/p) + 4{\cdot}((n/p) + (n/p^{2/3}))]$ in *bytes*, where n_{voxel} is the total number of voxels, p the number of processors, n the number of pixels in the final image; note that storage for boundary replication and rendering parameters are ignored.

After the rendering step, a partial image is produced on each processor. A parallel image compositing process then merges all partial images, in *depth order*, to achieve the complete image. This global combining process requires inter-processor communication. A direct compositing method [6] has been selected for the library design, which proceeds independently of the way in which data was partitioned. In the direct compositing method, the image plane is subdivided and each node is assigned a subset of the total image pixels. Each rendered pixel is sent directly to the node assigned that portion of the image plane. Processing nodes accumulate these partial image pixels in an array and composite them in proper order after all rendering is completed. Neumann [6] subdivided the image space in an interleaved fashion to ensure load balancing. Other parallel image compositing algorithms such as binary-swap developed previously by the author [3],

though superior in performance, would require regular data partitioning and the use of special data structures, thus less desirable for a generic PVR library design.

The scalability of the rendering phase is generally good since no communication is required. On the other hand, the scalability of the compositing part needs to be verified since there is some inter-processor communication involved. A performance analysis of the parallel compositing algorithm determines the communication cost for block data subdivision on a bidirectional torus to be $[c_1 \cdot p^{-(1/6)} \cdot n \cdot t_{trans} + c_2 \cdot p^{5/6} \cdot t_{overhead}]$ where c_1 and c_2 are positive constants smaller than 1, and t_{trans} is per-pixel transfer cost and $t_{overhead}$ is per-message overhead. Our analysis, consistent with Neumann's results [6], indicates that when message overhead is not high, the communication cost in fact decreases as the number of processors used increases. Thus the direct compositing method is acceptable.

At the end of image compositing, subimages are sent back to a remote workstation for storing or displaying. This results in a serial data stream, and message latencies may become a limiting factor with large number of processors. Image compression techniques can be used to achieve reasonable display rates. In our current implementation, direct transfer is used because most large-scale scientific simulations do not run multiple time steps per second, even on a massively parallel computer. In addition, when using a HiPPI frame buffer or Parallel I/O, the above problems become less severe.

Presently, the development of the library has been done on the Intel Paragon XP/S operated at the NASA Langley Research Center. The library has been written in C++ and the Intel's native communication library NX. Better portability can be achieved by moving to MPI or PVM. Finally, the library interface design includes both the API and the interactive user interface for setting up the renderer. The API should be as simple as possible. The application program passes the renderer the pointer to the subvolume, a record describing the geometry of the subvolume, and the rendering specifications. The interactive user interface should allow the user to set and change the rendering specifications including viewing and lighting parameters, image resolution, rendering rate and transfer functions.

4. A PARALLEL NAVIER-STOKES SOLVER

A parallel Navier-Stokes solver has been implemented particularly for testing the PVR library on the Intel Paragon. It allows us to experiment with different data distribution schemes and other design criteria. The unsteady compressible Navier-Stokes equations are a mixed set of hyperbolic-parabolic equations in time. We chose the classical MacCormack finite-difference method [4] for the solution of the equations. This explicit, two-step, three-time level scheme is second-order accurate in both time and space. During each time level, the first step (predictor) calculates a temporary "predicted" value, and the second step (corrector) determines the final "corrected" value. Forward and backward differencing are alternated between the predictor and corrector steps to avoid any bias due to one-sided differencing and to achieve second-order accuracy. The MacCormack method is used because it is easier to parallelize and implement. Later, for further testing of our rendering library, we will acquire more sophisticated CFD codes that are capable of simulating real-life problems.

The flow solver is implemented as a host-node model. As shown in Figure 1, the host program reads in the model specifications as well as the rendering specifications,

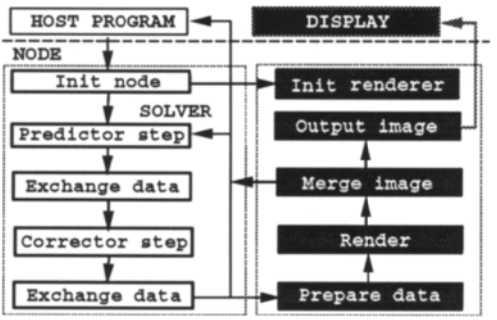


Figure 1. A Setting for Interactive Monitoring.

and broadcasts them to every node processor. If the data domain is subdivided into blocks, each block has six neighbors. At each time level, the solution values at the outermost boundary layers of each block are exchanged between nearest neighbors after each predictor and corrector step. *"Prepare Data"* is a routine written by the application programmer to calculate the flow properties to be visualized and store in the appropriate format for rendering.

5. TEST RESULTS

For testing, the simulation models laminar flow entering a rectangular region. Flow at 100 meters per second enters the small square inlet at the left of the rectangular domain. The region has one small square-inlets at one end and a fullwidth outlet at the other end. There are no body forces and external heat assumed. Figure 2 presents direct volume visualization of vorticity values of the overall simulated domain at selected time steps. To see the flow structures appearing in these images, one must adjust either the color or the opacity mapping to enhance a particular range of values of interest. For example, in Figure 2, regions of high vorticity are enhanced by mapping high vorticity values to red and higher opacity values, and low vorticity values to white and lower opacity values. The symmetric structures shown in all images verify the symmetrical boundary condition assigned. Three-dimensional volume visualization gives us a global, realistic view of the flow simulated, and thus a better understanding of the overall flow structures.

To evaluate the performance of both the simulation and the library, tests were performed on the Intel Paragon XP/S by using from eight to 216 compute nodes. The performance measures shown here were obtained by first calculating the average time over 300 time steps for three randomly picked viewing angles at each node; then the maximum time among the nodes is picked. Time is shown in seconds.

To see the scalability of both the solver and the rendering process, Table 1 shows time breakdown for rendering 256×256-pixel image on a fixed problem size of 60^3 data points using various numbers of processors. Figure 3 displays the same information graphically for revealing scalability by using log scale for both the x and y axes. As the timing results show, the simulation scales very well and thus achieves linear speedup. The rendering

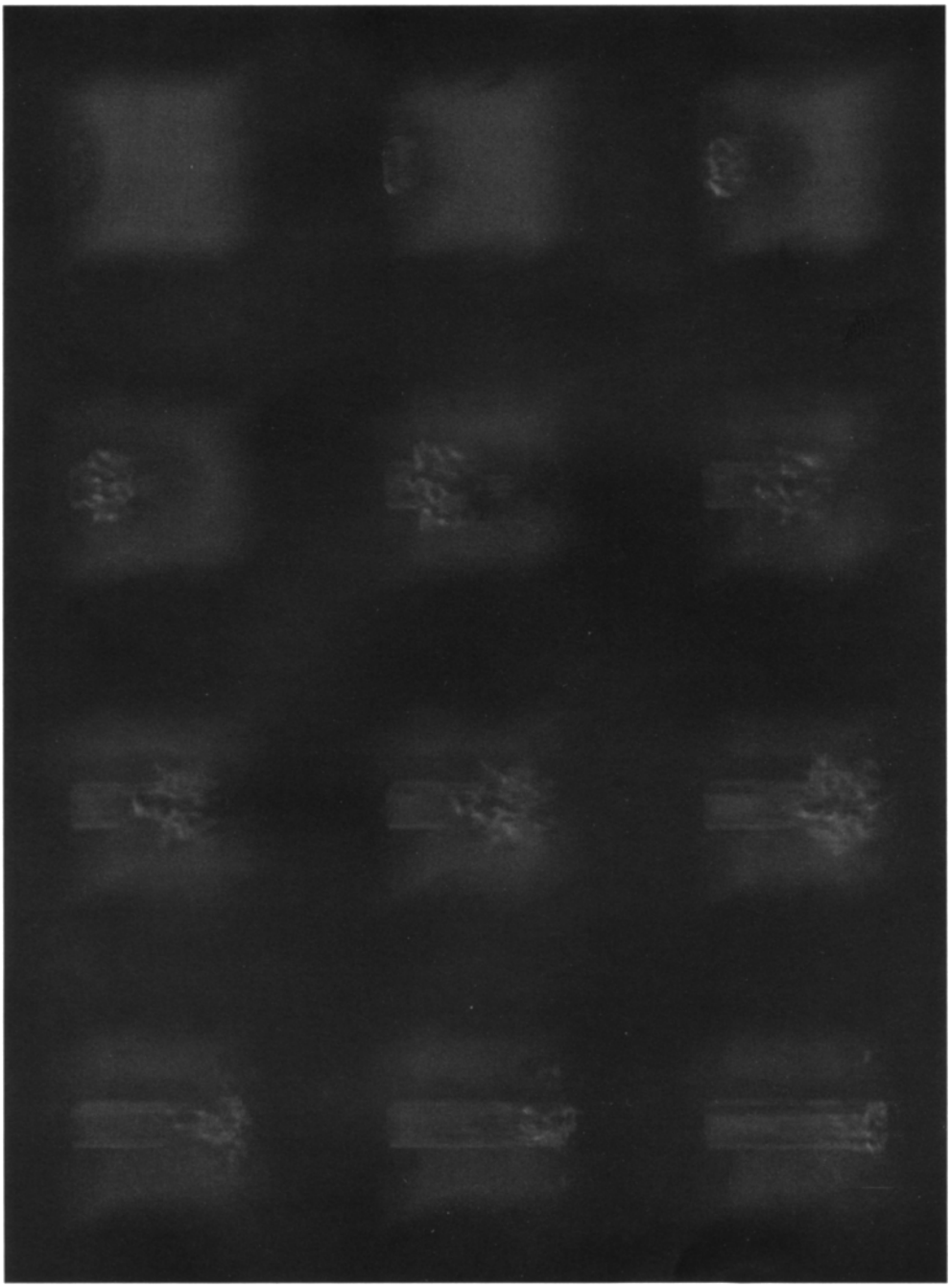


Figure 2. Tracking the Magnitude of Vorticity at Selected Time Steps.

Table 1
Time Breakdown for Using Different Number of Nodes.

node size	8	27	64	125	216
simulation (compute)	10.665	3.1435	1.3282	0.7542	0.4109
simulation (communication)	0.0453	0.0441	0.029	0.0246	0.0199
preparing data	0.3026	0.0963	0.051	0.0670	0.0433
resampling	13.01	4.21	1.877	1.102	0.681
compositing (raw compositing)	0.151	0.0525	0.0501	0.0455	0.0310
compositing (communication)	0.106	0.080	0.0672	0.0612	0.0583

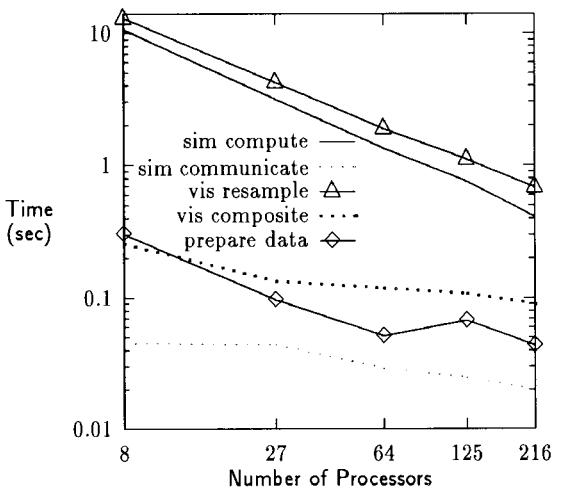


Figure 3. Timing for Both the Simulation and Runtime Visualization.

process scales well with the simulation as well as the parallel system. It also can be predicted that when running on a parallel system with faster processors and communication network, interactive steering of the simulation is feasible. Note that the time for image output and display is not shown here because our current display system is not representative.

Table 2 shows the time breakdown of one time step of the flow simulation and rendering 256×256-pixel image using 125 nodes. To show how the rendering process perform as the problem size increases the test were repeated for three different domain sizes: 60^3, 120^3 and 240^3. It is easy to see that simulation time dominates for typical problem sizes.

6. CONCLUSIONS

If the current trend in scientific computing continues, a parallel PVR library could be very useful to computational researchers who run their simulations on massively parallel

Table 2
Time Breakdown for Different Problem Sizes.

problem size	60^3	120^3	240^3
simulation (compute)	0.7542	5.6258	44.0
simulation (communication)	0.0246	0.073	0.254
preparing data	0.067	0.261	1.485
resampling	1.102	2.054	2.845
compositing (raw compositing)	0.0455	0.044	0.044
compositing (communication)	0.0612	0.085	0.0765

computers. Interactive visual responses reflecting simulation states and the physical phenomena modeled, allow better control of the simulation, and can offer additional insights into the physics behind the model. The parallel rendering library described in this paper is designed for distributed memory message passing environments. It provides an interface to handle volume data on a rectilinear grid. In CFD, rectilinear grids are still widely used but a majority of problems of more complex geometry require the use of either curvilinear or unstructured grids. This library can be extended to handle those grids using the algorithm described in [2]. Important future work includes implementing dynamic load balancing strategies and a graphics user interface and improving image I/O.

REFERENCES

1. Tom Crockett. Design Considerations For Parallel Graphics Libraries. Technical report, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA 23681-0001, June 1994. ICASE Report No. 94-49.
2. Kwan-Liu Ma. Parallel Volume Ray-Casting for Unstructured-Grid Data on Distributed-Memory Architectures. In *Proceedings 1995 Parallel Rendering Symposium*. ACM SIGGRAPH, October 1995.
3. Kwan-Liu Ma, J. Painter, C. Hansen, and M. Krogh. Parallel Volume Rendering Using Binary-Swap Compositing. *IEEE Computer Graphics and Applications*, 14(4):59–68, July 1994.
4. R.W. MacCormack. The Effect of Viscosity in Hypervelocity Impact Cratering. American Institute of Aeronautics and Astronautics, 1969. AIAA Paper No. 69-354.
5. Paul Mackerras and Brian Corrie. Exploiting Data Coherence to Improve Parallel Volume Rendering. *IEEE Parallel & Distributed Technology*, 2(2):8–16, 1994.
6. Ulrich Neumann. Parallel Volume-Rendering Algorithm performance on Mesh-Connected Multicomputers. In *Proceedings 1993 Parallel Rendering Symposium*, pages 97–104. ACM SIGGRAPH, October 1993.
7. Jason Nieh and Marc Levoy. Volume Rendering on Scalable Shared-Memory MIMD Architectures. In *1992 Workshop on Volume Visualization*, pages 17–24, 1992. Boston, October 19-20.
8. Claudio T. Silva and Arie E. Kaufman. Parallel Performance Measures for Volume Ray Casting. In *Proceedings Visualization '94*, pages 196–203, 1994.
9. Jianping Zhu. On the Implementation Issues of Domain Decomposition Algorithms for Parallel Computers. In *Proceedings Parallel CFD '92*, pages 427–438, 1992.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Integration of Particle Paths And Streamlines in a Spatially-Decomposed Computation

David Sujudi and Robert Haimes

Department of Aeronautics and Astronautics, MIT
Cambridge, MA 02139, USA

1. Introduction

The visualization of complex 3-D vector fields is difficult. To aid investigators in this effort, various visualization tools, such as instantaneous streamlines and particle paths, have been developed. These techniques are modeled on the experimental visualization techniques called streaklines and hydrogen bubbles, which have been very useful in examining boundary layer flows. Darmofal and Haimes [1] [2] have developed algorithms for computing streamlines and unsteady particle paths in 3-D vector fields, and implemented them in **Visual3**, a visualization software package developed in MIT's Computational Fluid Dynamics Lab [3].

However, the growth, in size and speed, of unsteady CFD calculations in the past few years demands a visualization software capable of handling huge 3-D unsteady data as well as providing tools for interrogating that data. To meet this goal, a parallel version of **Visual3**, named **pV3**, is under development at MIT [4]. **pV3** is designed for co-processing and also distributed computing. Co-processing allows the investigator to visualize the data as it is being computed by the solver. Distributed computing decomposes the computational domain into 2 or more sub-domains which can be processed across a network of workstation(s) and other types of compute engines. Thus, the processing needed by the visualization-tool algorithms (e.g., finding iso-surfaces, integrating particles and streamlines, etc.) can be done in parallel.

In a single-domain environment such as **Visual3**, a particle-path or streamline calculation stops when the integration hits the domain boundary. However, domain decomposition brings up concerns regarding where and how a particle-path or streamline integration can continue when the integration reaches a sub-domain boundary (also called internal boundary). This paper presents a scheme for managing the information movement needed to continue integrations across these internal boundaries.

In section 2, we explain how a computational domain is decomposed and the interactions between the processes involved in the CFD computation and the visualization. In section 3, we briefly describe the particle and streamline integration methods and discuss the issues encountered. Section 4 describes the solution and shows how it functions in a number of cases. Finally, the work is summarized in section 5.

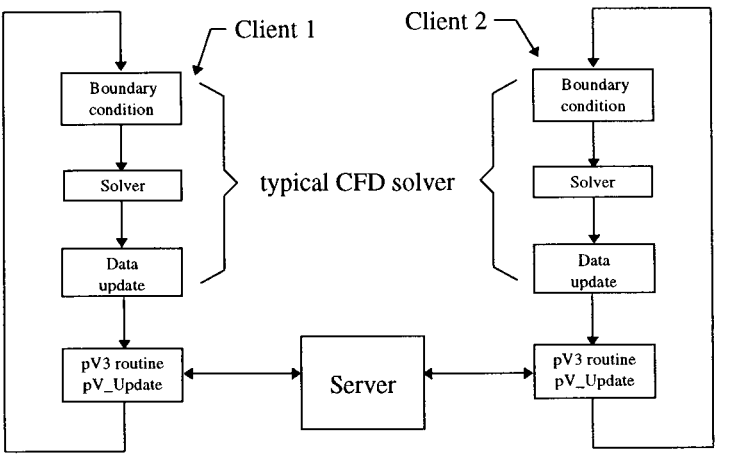


Fig. 1 Interaction between the server and clients in a typical 2-client setup.

2. Domain Decomposition

The CFD system decomposes the computational domain into sub-domains. The computational domain is made up of elements (such as tetrahedra, hexahedra, etc.), and the sub-domain boundaries are formed by the facets of these elements. **pV3** accepts any combination of these elements: disjointed cells (tetrahedra, pyramids, prisms, and hexahedra), poly-tetrahedral strips, and structured blocks. In **pV3**, the programmer/user can specify where an integration should continue (to a cell in a specific sub-domain, through a specific internal boundary in a sub-domain, or to try all sub-domains) when it reaches a sub-domain boundary.

Each sub-domain is handled by a separate process (called a client in **pV3**), and each computer in the distributed environment can execute one or more client(s). The user interface and graphic rendering are handled by a process (called the server) run on a graphics workstation. As an illustration, the interaction between the server and the clients in a typical 2-client setup is shown in Fig. 1. A call to a **pV3** subroutine (named **pV_Update**) is added to a typical CFD solver to handle the processing needed by **pV3**, including particle-path and streamline integrations. To keep the processing synchronized, each client does not exit **pV_Update** until the server transmits a time-frame-termination message. Only then can the clients continue to the next time step.

3. Particle-Path and Streamline Integration

A streamline is a curve in space which is everywhere tangent to the instantaneous velocity field. It is calculated by integrating:

$$\frac{d\vec{x}}{d\tau} = \vec{u}(\vec{x})$$

where $\vec{x}$ is the position vector, $\vec{u}$ the velocity vector, and τ the pseudo-time variable. The integration employs a fourth-order Runge Kutta method with variable pseudo-time stepping. The pseudo-time variable is used only for integrating streamlines and is independent of the actual computation time-step.

A particle path, on the other hand, represents the movement of a massless particle as time progresses. The calculation amounts to integrating:

$$\frac{d\vec{x}}{dt} = \vec{u}(\vec{x}, t)$$

The integration uses a fourth-order accurate (in time) backward differentiation formula. Details about the streamline and particle path integration algorithms can be found in [1] [2].

Both types of integrations start at a user-specified seed point and end when a computational boundary is reached. In between, the integration might pass through sub-domain boundaries. When this occurs, information needed in continuing the integration (referred to as a "continue-integration request" or "transfer request") must be sent to other client(s). The information usually comprises of integration state, identification number for the particle/streamline, rendering options, client identification, etc. The client(s) receiving the transfer request has the option to do one of the following:

- accept the transfer: determines that the integration continues and ends in its sub-domain.
- request a re-transfer: determines that the integration should try to continue in one or more other client(s).
- reject the transfer: determines that the integration does not continue in its sub-domain and ignore the request.

The algorithm for determining whether an integration hits an internal boundary, or whether a client accepts/rejects a transfer, or requests a re-transfer involves checking the spatial location used in the current integration stage against the sub-domain space. The details will not be discussed in this paper, but can be found in [1]. We will concentrate on the problems encountered in managing transfer requests, and on how the client(s) and the server must interact to ensure that integrations are continued, while keeping the process as efficient as possible. The issues can be stated by the following questions:

- Due to the existence of a "try all sub-domain" specification, how do we avoid repeatedly sending the same request to the same client(s)? This is an efficiency issue.
- How can the server know when to safely send the time-frame-termination message? Sending this message when there is a possibility of further integration transfers could prematurely abort the integration of some particles or streamlines. Thus, the server must know when all transfers are complete for the current time step. Otherwise, the time-frame-termination signal can not be sent and the clients (involving the CFD solver) will remain in a wait state, stalling the entire calculation. This situation is clearly undesirable.
- How do we maximize the use of the parallel environment in integrating streamlines and particle paths? These integrations are essentially a serial process. The integration starts at some point at the beginning of the time step and stops at another point at the end of the time step. In between, the process must be done serially. In **pV3**'s parallel environment, we are able to distribute the work of integrating streamlines/particles because each client

processes only the objects in its sub-domain. However, problems in optimally exploiting the distributed environment can occur due to the need to continue integration across internal boundaries and also the existence of other types of requests that the clients need to handle. We can foresee cases where a mixture of transfer requests and other requests could generate a load imbalance between the clients. This issue will be illustrated and made clearer when we discuss the solution in the next section.

When it comes time for rendering, each client sends the server the relevant information about all the particles (i.e., their locations) that end up in its sub-domain at the end of the current time step. A streamline, however, is divided into segments. A new segment begins every time a streamline integration crosses an internal boundary. To render a streamline, each client sends the server information about the segments (e.g., the points forming the segment) within its sub-domain. The server combines the segments, constructs a complete streamline and then renders it.

Although, the underlying construction of a particle and a streamline are completely different, the issues encountered in continuing their integration across an internal boundary are, in fact, similar. For a particle, the question is where it should end up at the end of the current time step. For a streamline, the question is where its next segment should begin at the end of the current pseudo-time step. We will take advantage of this similarity in developing a scheme that is usable in both cases. The evolution of this scheme will be described in the next section.

4. Solution

First, we propose the following scheme:

1. We have decided to process transfer request (TR) through the server instead of having the clients communicate directly. Thus, to continue the integration of a particle/streamline, a client sends a TR to the server, which then distributes it to one or more client(s), depending on the particular internal-boundary specification. We believe this solution gives us a simpler and cleaner system overall because all other visualization message traffic is client-server and only one process (the server) needs to manage the transfer requests.
2. For each particle/streamline segment that requires transfer across internal boundary, the server keeps a list of clients to which the TR have been sent. The server also assigns a unique identification number (denoted by TID, for "transfer id") for each of these particles/streamline segments.
3. To prevent sending the same TR to a client, the server can send a TR for a TID to a particular client only if the client is not already in the client list. Thus, to avoid sending a transfer request back to the originating client (i.e., the client which initially sends the transfer request), the originating client must automatically be logged in the client list.
4. Transfer requests have higher priorities (at the client and the server) over other types of visualization requests. Thus, if the request queue of a client or the server contains a TR, that request will be handled first. In most situations, this procedure will help lessen load imbalances. As an example, consider a 3-client setup where the request queue of client 2 contains R_1, R_2, and lastly a transfer request TR, while that of client 1 and 3 contains only R_1 and R_2. R_1 and R_2 could be requests to calculate iso-surfaces, find swirling flow,

generate cut planes, etc. Let's suppose that TR will generate a re-transfer to client 3. Requests R_1 and R_2 could take much longer to process in client 2 than in client 3, depending on the partitioning, type of request, flow conditions, etc. If the clients handle the requests according to queue order, client 3 might finish before client 2. Then, client 3 will have to wait until client 2 handles the transfer request before it can process the re-transfer that will be generated by that request. On the other hand, if TR has higher priority and is processed first by client 2, client 3 will receive the re-transfer sooner. The entire process will take less wall-clock time than without prioritizing.

5. After a client processes a TR - whether accepting, rejecting, or re-transferring the request – it sends a transfer – processed acknowledgement (TPA) back to the server. Since the originating client is automatically logged in the client list, a TPA is also automatically associated with the originating client.
6. The transfer process for a particular TID is completed when the server receives TPAs from all the clients to which the TID has been transferred. To simplify the algorithm for checking transfer completion we must ensure that when a client sends a TR and then a TPA (such as in a re-transfer request), the server receives them in the same order. Then, the server will record the re-transfer (if any is allowed) before the TPA. This can be accomplished by setting TPA's priority to be the same as TR's. Now, determining transfer completion can be done at any moment by identifying those TIDs whose client list has a complete set of TPAs. When all TIDs have a matching set of clients and TPAs, the server knows that no more transfers will be requested during the current time step.

To illustrate this scheme, we are going to show a 3-client example in which 3 particles require transfers in the current time-step and 1 particle does not, as shown in Fig. 2. Note that no transfer will be requested for the fourth particle (the one in client 3) because it stays within client number 3. Due to the similarity in continuing particle and streamline integration, we will show examples only for particles. Shown in a step-by-step manner, the scheme works as follows (at each step, the particle transfer log for each TID is summarized in the corresponding table):

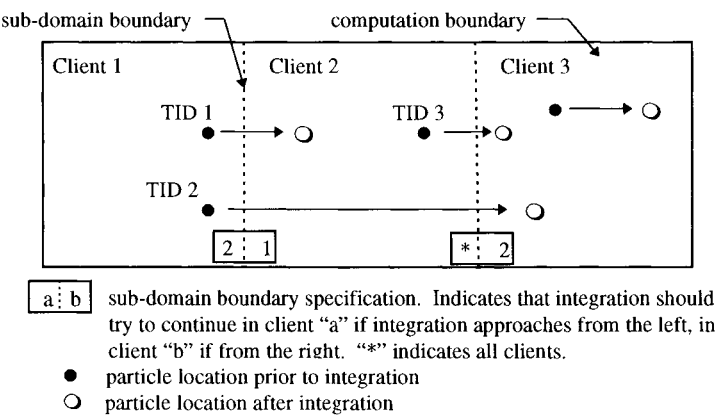
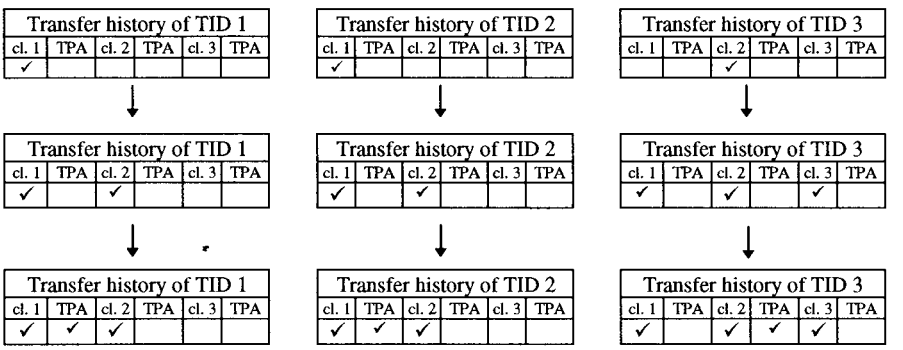
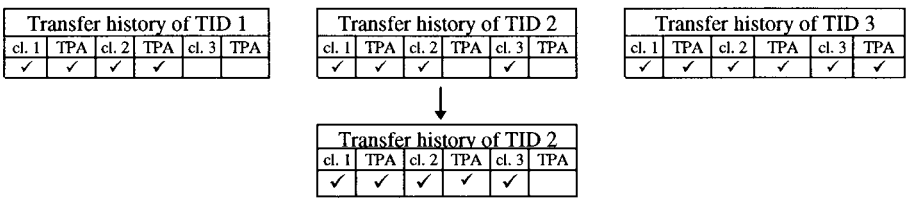


Fig. 2 Illustration for a 3-client example

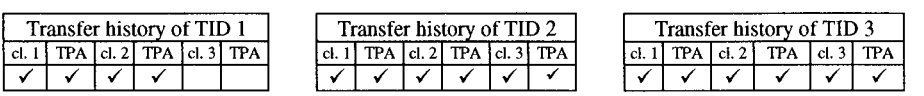
1. The server receives two transfer requests from client 1 and one from client 2, and assigns them TID 1, 2, and 3 as shown in Fig. 2. In the tables below, a "✓" under client number indicates that a TR has been sent to that client (or the client is the originating client), and a "✓" under TPA means that the server has received a TPA from the corresponding client. Complying with the internal-boundary specification, the server transfers TID 1 and 2 to client 2, and attempts to transfer TID 3 to client 1, 2, and 3. However, since TID 3 already has client 2 in its transfer list, TID 3 will not be transferred there again. Note that a TPA is automatically assigned to the originating client.



2. Client 1 will reject TID 3 and send a TPA. Client 2 will accept TID 1 and send a TPA. Client 2 will also request that TID 2 be re-transferred to all other clients, and then send a TPA for TID 2. However, since client 1 and 2 are already in the client list of TID 2, TID 2 will only be transferred to client 3. Client 3 will accept TID 3 and send a TPA. At the end of this step the transfer process for TID 1 and 3 is complete because each client in their list is paired with a corresponding TPA.



3. Client 3 accepts TID 2 and sends a TPA. At this point (and only at this point), all TID have a complete list of TPAs (i.e., each client in the list is paired with a TPA). Thus, all transfers are complete for this time step.



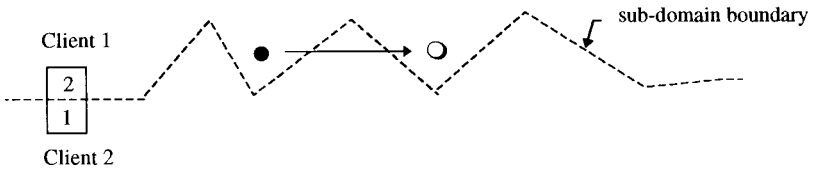


Fig. 3 Integration of a particle requiring a transfer from client 1 to client 2 and a re-transfer from 2 back to 1.

Now consider the situation illustrated in Fig. 3. Initially, client 1 transfers the particle to client 2, then client 2 requests a re-transfer back to client 1. However, since the particle originates from client 1, the above scheme will disallow this, and the integration stops. To take care of cases such as this, we modify the above scheme to allow a particle (or streamline segment) to be transferred to the same client twice. Since multiple transfers to the same client are now allowed, there is no need to automatically include the originating client in the client list. For the situation in Fig. 3, the process goes as follows:

1. The server receives a transfer request from client 1, and transfer the integration to client 2. In the table, we now have 2 columns under client and TPA to reflect that the particle can be transferred to the same client twice.

Transfer history of TID 1							
Client 1		TPA		Client 2		TPA	
				✓			

2. Client 2 requests that the particle be re-transferred to client 1, and then sends a TPA.

Transfer history of TID 1							
Client 1		TPA		Client 2		TPA	
✓				✓		✓	

3. Client 1 accepts the transfer and sends a TPA. Now we have complete pairings of the check mark, indicating the transfer process for the current time step is complete.

Transfer history of TID 1							
Client 1		TPA		Client 2		TPA	
✓		✓		✓		✓	

This solution, however, is not without trade-offs and limitations. In cases where re-transfers to the same client are not needed (such as in the first example above), unnecessary multiple transfers to a client might be made. In some situations, a client might accept a transfer twice, creating 2 instances of the same particle (or streamline segment). To prevent this, every time a client receives a transfer request it checks whether it has previously accepted the object. If it has, the request is ignored. The checking is done by comparing the global identification number, which is unique through the duration of the visualization session, or the TID, which is unique during each time step

This solution is also not a general one. Consider the situation shown in Fig. 4, in which the particle needs 3 transfers to client 2. This will not be possible without increasing the maximum number of transfers to 3, which will further reduce the efficiency of the scheme. We

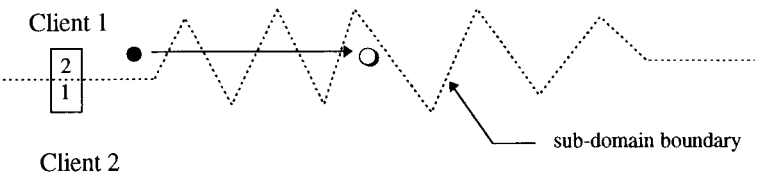


Fig. 4 Illustration of a particle requiring 3 transfers to the client 2

can see that however large this number is set to, the solution will never be general. Thus, as a trade-off between efficiency and more generality we limit the number of transfers to 2. We believe setting the maximum to 2 transfers covers most cases without significantly compromising efficiency. Should a particle-path integration require more than two visits to the same client, the integration will abort and the particle will be "lost". If it is a streamline integration, only part of the streamline will be rendered (from the seed point to the last point before the unsuccessful integration transfer). Fortunately, this limit is even less of a problem in streamline integration because the algorithm uses a pseudo-time step limiter which is based on the cell size as well as local vector field data. This limiter insures that the next requested point in the integration is no more than the cell's size from the current position. If the size of the neighboring elements do not change drastically, the problems posed by this scheme will be minimal.

5 Summary

We have presented a method for continuing integrations of streamline and particle path across sub-domain boundaries. Although the underlying structures of streamline (a curve) and particle (a point) are completely different, we have exploited the similarity in continuing their integrations across internal boundaries to develop a scheme that works in both cases. The scheme manages the flow of information between the clients, where the integrations are done, and the server, which does the rendering, in order to minimize network traffic, avoid multiple instances of the same object, and make efficient use of the parallel environment. It also provides a simple test to determine whether more integration transfers will be done within the current time-step. This knowledge enables the server to know when it can safely instruct the clients to proceed to the next time step.

References

[1] D. Darmofal and R. Haimes, "Visualization of 3-D Vector Fields: Variations on a Stream," AIAA Paper 92-0074, 1992.

[2] D. Darmofal and R. Haimes, "An Analysis of 3-D Particle Path Integration Algorithms for Unsteady Data," submitted to the AIAA CFD Conference, June, 1995.

[3] R. Haimes and M. Giles, "**Visual3**: Interactive Unsteady Unstructured Visualization for CFD," Computing Systems in Engineering, 1(1):51-62, 1990.

[4] Haimes, "**pV3**: A Distributed System for Large-Scale Unsteady CFD Visualization," AIAA Paper 94-0321, 1994.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

Interactive Volume Rendering on Clusters of Shared-Memory Multiprocessors*

Michael E. Palmer[1,3], Stephen Taylor[1], and Brian Totty[2]

[1]Mail Code 256-80, California Institute of Technology, Pasadena, CA 91125 USA
[2]Mail Stop 580, Silicon Graphics Computer Systems, Mountain View, CA 94043 USA

[3]mep@scp.caltech.edu, http://www.scp.caltech.edu/~mep

We describe methods for parallel partitioning and load balancing of volume rendering on a cluster of shared-memory multiprocessors; good load balance and parallel speedup are demonstrated. We describe a method of dynamic resolution reduction and image refinement which typically allows a four-fold performance increase.

1. CONTRIBUTIONS

We present several methods to facilitate interactive volume rendering of very large datasets (512^3 to 1024^3 elements) at very high resolution (3200 by 2400 pixels) on several interconnected shared-memory multiprocessors. We describe an image-based partitioning method which is suited to dynamic load balancing; it uses measurements of the first moment of the distribution of work to accurately balance workload. The partitioning method is suited to division of labor between several shared-memory multiprocessors, each rendering a portion of the screen, without communication of volume elements between them. Our dynamic resolution reduction and image refinement method is particular useful on very high resolution displays; the method reduces resolution on the edges of the screen by the amount required to maintain a requested frame rate; the image is iteratively refined to full resolution when the viewpoint is stationary.

These techniques were demonstrated at the Silicon Graphics, Inc. vendor booth at *Supercomputing '94*, in Washington D.C., November, 1994, in conjunction with the Army High Performance Computing Research Center at the University of Minnesota. The rendering engine was a POWER CHALLENGEarray of eight nodes, with a total of 84 R8000 processors, which send finished images over HiPPI for display by two POWER Onyx machines, each with two Reality Engine^2 graphics boards.

2. HARDWARE CONFIGURATION

In the center of Figure 1 are the two POWER Onyx nodes; each contains eight R8000 processors, 2GB of shared memory, 300GB of disk, and two Reality Engine^2 graphics

*This research is sponsored by the Department of Defense, AASERT award number N0014-93-1-0843, under a parent grant from the Advanced Research Projects Agency, ARPA Order 8176, monitored by the Office of Naval Research under contract N00014-91-J-1986.

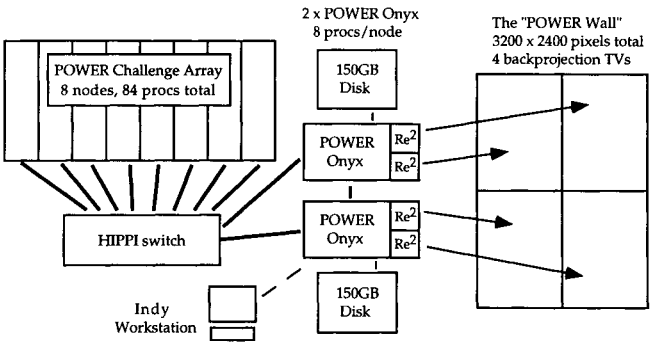


Figure 1. Hardware Configuration for *Supercomputing '94*

boards. Each of the four Reality Engines drives a projection TV display at a resolution of 1600 by 1200 pixels. The four TV projectors are aligned to display one large image of 3200 by 2400 pixels.

To the left of the figure, connected by two HiPPI links to the Onyx machines, is the POWER CHALLENGEarray; this consists of eight nodes, with 2GB of RAM per node, a total of 84 R8000 processors, and no graphics hardware. The Challenge Array nodes are connected to each other and to the Onyxes by HiPPI. Memory is not shared between the nodes. An Indy workstation is used to run the graphical user interface. Control messages are transmitted by ethernet between the machines.

3. RELATED WORK

Volume rendering is replete with parallelism. There are two classes of parallel partitioning methods generally used for volume rendering. The first class, *image partition* [6,9,8], assigns subregions of the image space to processors. Processors then interact with the subsets of the data volume that affect their image subregions. The second partitioning class, *object partition* [3,7,2,4,11], assigns subvolumes of the data to processors. Processors compose the partial effects of their subvolumes of data back onto the image plane.

Methods of spatial subdivision can be divided into those which employ a uniform, static subdivision [2,5], and those that divide space in an adaptive manner, either by iterative [7,3,9], or recursive [4,11] subdivision methods. Adaptive subdivisions may respond to changing conditions of workload during execution [3,7,9] to balance load among processors.

4. PARTITIONING ON A CLUSTER OF SHARED MEMORY MULTIPROCESSORS

The image partitioning method we chose is designed to accommodate dynamic load balancing by allowing flexible reassignment of processor power to emerging concentrations of work load. Lines dividing the processors are shifted to reassign work.

The method, illustrated in Figure 2, divides the entire screen by a $\log N$ recursive binary cuts, where N is the total number of processors. The cuts alternate in the X and Y directions. The first, and subsequent odd divisions, are parallel to the Y axis; the second, and subsequent even divisions, are parallel to the X axis, as illustrated in Figure 2.

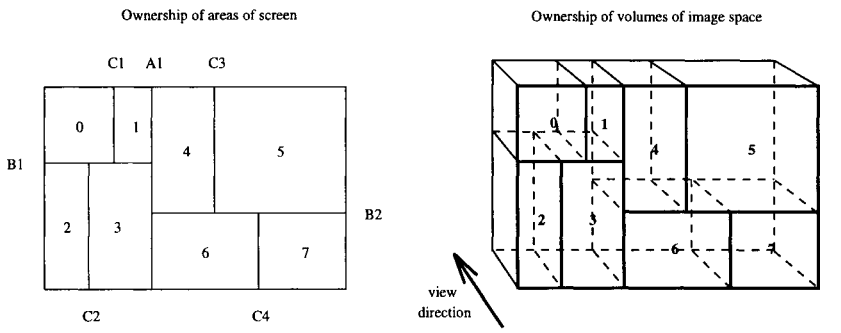


Figure 2. Subdivision of Screen Area and Image Space

The POWER CHALLENGEarray is our rendering engine; we do not use any special graphics hardware for rendering. The array nodes send pieces of finished frames over HiPPI to the Onyx machines for assembly into complete frames and display. Since, in our hardware configuration, each of the two Onyx machines contains the graphics hardware to display to one half of the screen, it greatly simplified communication to assign one half of the POWER CHALLENGEarray to each Onyx, and to have each half of the Array generate one half of the screen image. The division between the two Onyxes is along line A1 in the figure. The drawback to this approach is that fixing line A1 is a contraint on load balancing. (For a single display configuration we do allow this line to move.)

Designers of parallel algorithms can often trade memory costs for communication costs. In our case, we had more than enough memory to store large (hundreds of megabytes) datasets — 2GB per node, and wanted to avoid communication of large volumes of the dataset; we therefore replicate the entire dataset onto each array node.

5. LOAD BALANCING WITH FIRST MOMENT VECTORS

We present a novel means to measure the distribution of work in image space using the *first moment of the workload distribution*[10]. As rays pass through the data volume, they intersect voxels of data; we count each ray-voxel intersection as a unit of work. We find the first moment of these points in three-space to estimate the distribution of work. The partitioning for the next frame is based on this distribution for the current frame. The method therefore relies on a form of *temporal coherence* [1] — that the viewpoint for one

rendered view will be similar to the last, by virtue of smooth head motion.

Using the first moment of these intersections, instead of the common alternative of simply *counting the intersections* in each area of the screen, allows a qualitatively superior load balance: first, because a first moment measurement of a region more accurately describes the density distribution than a simple count for that region, it allows more accurate determination of ideal load balance for motionless data; furthermore, because the first moment is a vector, it can be matrix-multiplied by the difference between the last and current view matrices. This effectively *anticipates to the next frame* the change in load density incurred by data movement, so that moving data can also be precisely balanced, without a chronic time-lag.

Load balancing is achieved by the movement of the lines dividing areas of the display screen, shown in Figure 2, appropriately for new distributions of workload. We place line $A1$, the central line, on the X coordinate of the first moment of the entire dataset; we next place lines $B1$ and $B2$ on the Y coordinates of the first moments of the data on the left and right sides of line $A1$, respectively; and so on.

6. DYNAMIC RESOLUTION REDUCTION

In volume rendering, there is a trade off between high resolution and high frame rate. By a technique of dynamic resolution reduction, we can automatically choose which of these is most important to the user. The pattern of resolution reduction is shown in Figure 3. In the center of the screen, we maintain full resolution. Around this central area, we place concentric bands of reduced resolution. When the user stops moving, we refine the image in several steps until the entire screen is at full resolution, as shown from left to right in the figure. With this technique, we obtained a typical four-fold reduction in total number of rays cast, and a corresponding four-fold frame rate increase.

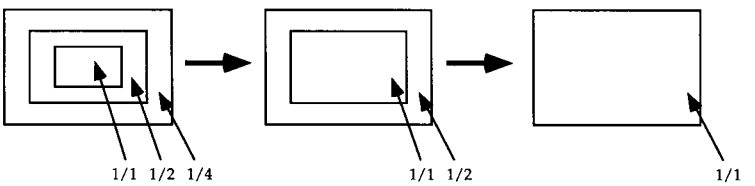


Figure 3. Dynamic Resolution Refinement for Higher Frame Rate

7. RESULTS

Our experiments consist of timings of a smooth head movement around a 375MB version of the Visible Human dataset [12] at a resolution of 640x486 pixels. The head movement consisted of an orbiting motion around the human figure in the dataset, combined with

an in and out zoom, as shown in the sequence of frames in Color Plate 1. In the color plate, the white squares overlaid on the data show the areas of the screen assigned to each of sixteen processors.

Figures 4 and 5 show several representations of the same set of experiments. The left side of Figure 4 shows raw frame times for 1, 2, 4, 8, and 16 processors on a single array node. The left-right symmetry of the plot is due to the symmetrical orbit and in-out zoom of the head motion. The right side of the figure shows parallel speedup over a single processor: two and four processors get excellent speedup. Eight and sixteen processors get average speedups of seven and thirteen respectively.

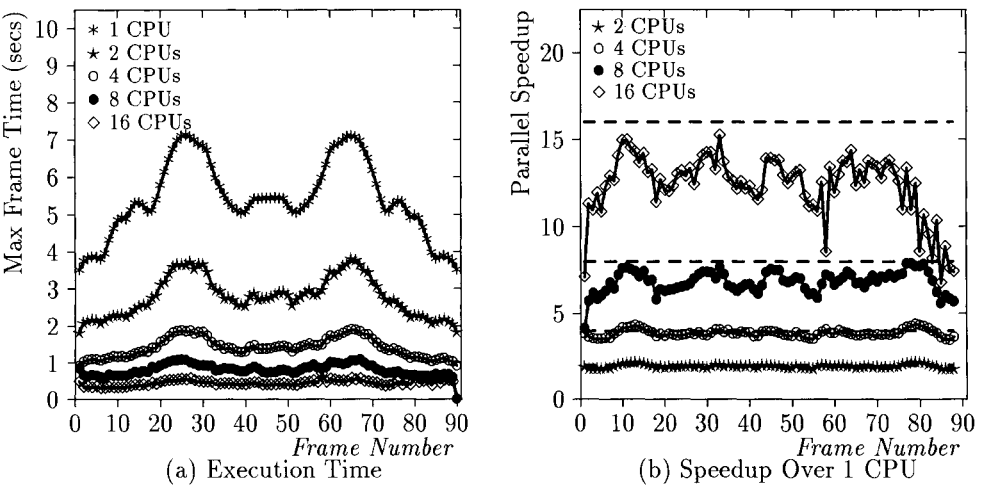


Figure 4. Execution Time and Speedup of Image Partition (Test Suite 0)

Figure 5 shows load balance on 2, 4, 8, and 16 processors. The measure shown is the frame time for the average processor divided by the time for the slowest processor, multiplied by 100. If these times are equal, then all processors finish at the same time, and load balance is perfect. Two and four nodes get excellent load balance; eight nodes get between eighty and ninety percent balance. Sixteen nodes get between seventy and eighty percent balance for much of the experiment, but between frames 80 and 90 get only sixty percent. The poor parallel speedup for sixteen nodes between frames 80 and 90 on the right side of Figure 4 may be attributed to the load balance during these frames.

Recall that load balance for one frame depends on the nearness of the viewpoint to that of the last frame. As the number of processors increases, the area of the screen assigned to each processor becomes smaller; line placement is therefore more sensitive to differences in viewpoint between the last and current frames. This experiment forced a fixed amount

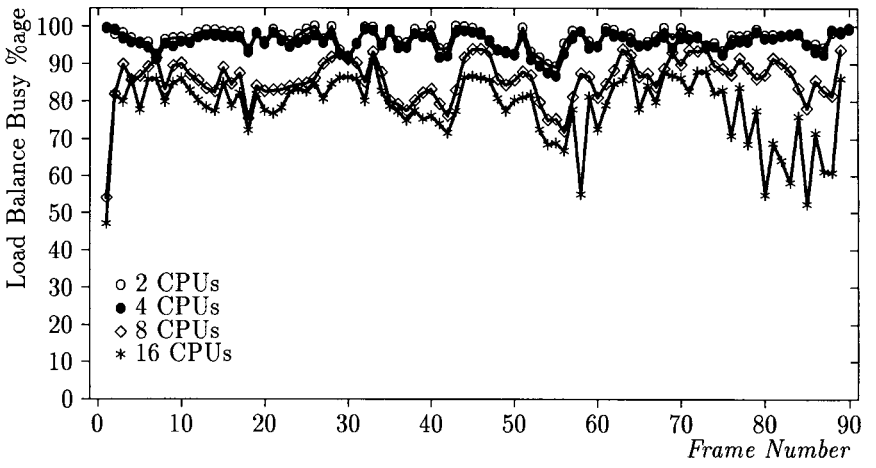


Figure 5. Load Balance of Image Partition (Test Suite 0)

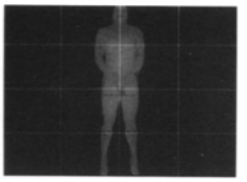
of head motion *per frame* on the processors. In practice, a machine with more processors will run faster, while head motion demands *per second* would remain constant, yielding a nearly constant amount of head motion per frame as number of processors increases, and correspondingly better load balance.

The above results were taken on a single node of sixteen processors; using two nodes of eight processors, we found a parallel speedup of approximately 1.75 over one node of eight processors, when line A1 (from Figure 2) was allowed to move. On four nodes of eight processors we found a parallel speedup of 3.0 over one node, yielding a frame rate of between 2 and 3 Hz. As with the above experiments, this was while forcing a fixed amount of head motion *per frame* on the processors. We are currently conducting further work on multi-node arrays, which will appear in a subsequent paper.

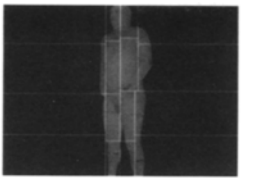
All experiments cited were run without dynamic resolution reduction and image refinement; adding this technique reduces the total number of rays cast by a factor of four, yielding a four-fold improvement in performance.

8. CONCLUSIONS

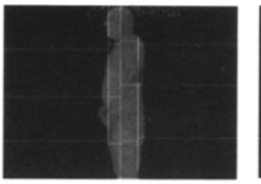
We have presented several methods for interactive volume rendering on a cluster of shared memory multiprocessors. We have described a partitioning method suited to dynamic load balancing, which maps on to such a hardware configuration. We have described a method to accurately estimate workload distribution with first moment vectors, and how to dynamically load balance using this information. We have practically demonstrated good load balance and scaling. We also presented two related methods which are particularly relevant to rendering final images at very high resolution: a method to reduce the



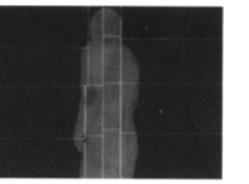
frame 0



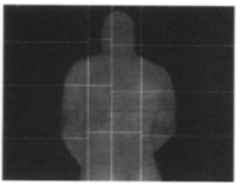
frame 5



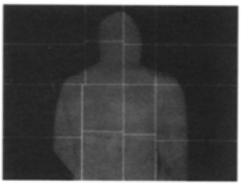
frame 10



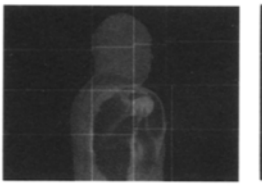
frame 15



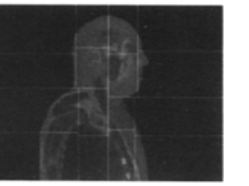
frame 20



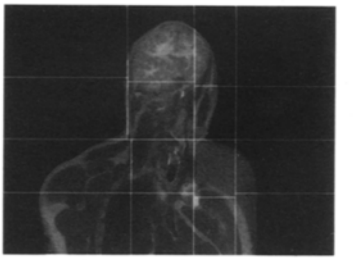
frame 25



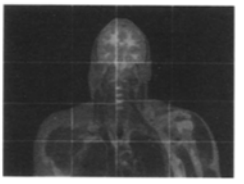
frame 30



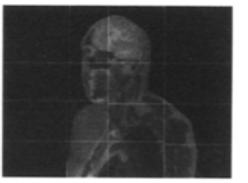
frame 35



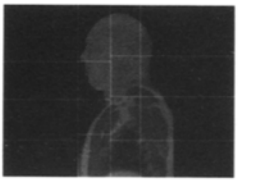
frame 40 (large)



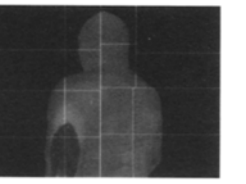
frame 45



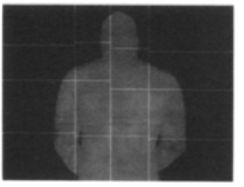
frame 50



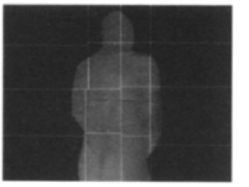
frame 55



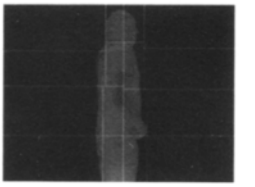
frame 60



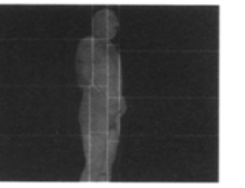
frame 65



frame 70



frame 75



frame 80

Color Plate 1: Selected Frames from Test Suite 0, Visible Human dataset

total number of rays cast, which maintains high resolution at the center of screen; and a method to efficiently refine the image to full resolution when the viewpoint becomes stationary.

9. ACKNOWLEDGEMENTS

The authors would like to thank Steve Anderson, Tom Ruwart, and Paul Woodward of the Army High Performance Computing Research Center at the University of Minnesota for giving us the opportunity to contribute to the AHPCRC's POWER Wall Project.

REFERENCES

1. S. Badt, Jr. "Two Algorithms for Taking Advantage of Temporal Coherence in Ray Tracing." *The Visual Computer* (1988), 3:123-131, Springer-Verlag, 1988.
2. J.G. Cleary, B.M. Wyvill, G.M. Birtwistle, R. Vatti. "Multiprocessor Ray Tracing." *Computer Graphics Forum*, 5:3-12. North-Holland. 1986.
3. M. Dippé, J. Swensen. "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis." *ACM Computer Graphics* 18(**3**):149-158, 1984.
4. H. Kobayashi, T. Nakmura, Y. Shigei. "Parallel processing of an object space for image synthesis using ray tracing." *The Visual Computer* (1987), 3:13-22, Springer-Verlag 1987.
5. H. Kobayashi, S. Nishimura, K. Kubota, T. Nakamura, Y. Shigei. "Load balancing strategies for a parallel ray-tracing system based on constant subdivision". *The Visual Computer* (1988), 4:197-209, Springer-Verlag 1988.
6. M. Levoy. "Design for a real-time high-quality volume rendering workstation." *Chapel Hill Workshiop on Volume Visualization*, May 1989:85-92.
7. K. Nemoto, T. Omachie. "An adaptive subdivision by sliding boundary surfaces for fast ray tracing." In: *Proc Graphics Interface '86, Canadian Information Processing Society*, pp. 43-48.
8. U. Neumann. *Volume Reconstruction and Parallel Rendering Algorithms: A Comparative Analysis.* Ph.D. Thesis, Dept. of Computer Science, University of North Carolina, 1993.
9. J. Nieh, M. Levoy. "Volume rendering on scalable shared-memory MIMD architectures." *ACM SIGGRAPH 1992 Workshop on Volume Visualization*, pp. 17-24.
10. M.E. Palmer. *Immersing the Scientist in Data: Interactive Visualization of Unstructured Scientific Data on Concurrent Architectures.* Master's Thesis, Dept. of Computer Science, California Institute of Technology, Pasadena, CA. Caltech-CS-TR-94-06. 1994.
11. T. Priol, K. Bouatouch. "Static load balancing for a parallel ray tracing on a MIMD hypercube." *The Visual Computer* (1989), 5:109-119, Springer-Verlag 1989.
12. The Visible Human dataset is publicly available from Michael J. Ackerman, the Visible Human Project, National Library of Medicine.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Application of Parallel Multigrid Methods to Unsteady Flow: A Performance Evaluation

A. T. Degani[a*] and G. C. Fox[†]

[a]Northeast Parallel Architectures Center
Syracuse University
Syracuse, NY 13244, USA

ABSTRACT

The parallel multigrid time-accurate calculation of the unsteady incompressible Navier-Stokes equations is carried out using both explicit and implicit schemes. In the explicit solution method, a 'lumped' scheme is employed at the coarsest multigrid levels where all the processors solve the same problem. On the other hand, in the implicit method, in which the equations are solved in a fully-coupled mode, a 'semi-distributive' scheme is used where the effective number of active processors decreases logarithmically with each coarsening of the mesh at the coarsest levels. Both 'V' and 'W' cycles are implemented in the explicit method and the convergence rates and execution times are compared. It is demonstrated that good speedups are obtained for the implicit scheme and the slight degradation in parallel efficiency, relative to calculations performed on the finest grid, is dominated by increased convergence rates.

1. INTRODUCTION

Parallel multigrid computation offers two desirable properties for the solution of large problems: i) computational effort scales with problem size, typically of $O(N \log N)$ where N denotes the size of the problem, and ii) implementation is scalable on coarse-grained distributed machines; specifically, good speedups and parallel efficiencies are possible as the number of processors p increases for N/p large and fixed. Here the parallel implementation of the multigrid method is applied to the time-accurate calculation of the unsteady incompressible Navier-Stokes equations. The primary objective here is the evaluation of parallel multigrid methods to unsteady flow in terms of convergence rates and parallel efficiency as the size of the problem and number of processors is varied. As a first step, a regular structured two-dimensional computational domain is considered; however, since a primitive-variable formulation is adopted, an extension to three dimensions is straightforward. Both explicit and implicit schemes on a staggered mesh are considered, and, in the former, the relative effectiveness of the 'V' and 'W' multigrid cycling algorithms is

*Alex G. Nason Research Fellow
†Director, Professor of Computer Science and Physics

evaluated.

The multigrid algorithm was first coded on a uniprocessor machine in FORTRAN 77 but designed in such a fashion so as to allow the subsequent seamless transition to the development of a Single Program Multiple Data (SPMD) code with message passing. The results reported here were obtained on a 32-node CM-5 installed at NPAC. The data are distributed in a block-block layout on a two-dimensional mesh of abstract processors and, for efficient memory utilization, the local data in each processor are mapped onto a local one-dimensional array. The array of cells in each processor is augmented by a buffered boundary of one cell thickness at all multigrid levels where data from neighboring processors are stored.

The unsteady incompressible Navier-Stokes equations are given by

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \mathbf{g} - \nabla p + \frac{1}{Re} \nabla^2 \mathbf{u}, \qquad \nabla \cdot \mathbf{u} = 0, \tag{1}$$

where the equations are nondimensionalized by appropriate velocity and length scales, U_o and L_o, respectively, and the kinematic viscosity ν. Re is the Reynolds number defined as $Re = U_o L_o / \nu$.

2. EXPLICIT SCHEME

The explicit scheme adopted here is the projection method [1] in which the momentum equation is split according to

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} + \mathbf{u}^n \cdot \nabla \mathbf{u}^n = \mathbf{g}^n + \frac{1}{Re} \nabla^2 \mathbf{u}^n, \tag{2}$$

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^*}{\Delta t} = -\nabla p^{n+1}, \tag{3}$$

where the superscript n denotes the evaluation of a quantity at time level n and $\mathbf{u}^*$ is an intermediate provisional value of the velocity field; this scheme is formally $O(\Delta t)$ accurate. Upon taking the divergence of equation 3 and using the continuity equation evaluated at time level $n+1$,

$$\nabla^2 p^{n+1} = \frac{1}{\Delta t} \nabla \cdot \mathbf{u}^*. \tag{4}$$

The boundary condition for the above Poisson equation for the pressure is obtained by projecting equation 3 on the boundary Γ of the computational domain (i.e. a dot product of equation 3 and the normal vector $\hat{\mathbf{N}}$ of Γ) and letting $u_\Gamma^{n+1} = u_\Gamma^*$ [1]. This yields

$$\left(\frac{\partial p}{\partial N} \right)_\Gamma^{n+1} = 0. \tag{5}$$

Note that the homogeneity of the Neumann boundary condition above is not physical but only a numerical artifice made possible due to the use of a staggered grid [1]. In the projection method outlined above, time-stepping is accomplished in the following manner: *(i)* the provisional value of the velocity $\mathbf{u}^*$ is obtained from equation 2, *(ii)* the

Poisson equation 4 is solved for the pressure subject to the boundary condition given by equation 5, and, finally, *(iii)* the velocity field at time level $n+1$ is obtained from equation 3.

The projection method is implemented on a staggered mesh where the normal velocities are defined at the midpoint of the cell faces and the pressure is defined at the cell center. All spatial derivatives are evaluated using second-order central differences. Since the no-slip condition cannot be satisfied exactly on a staggered mesh, fictitious points are defined along the boundary of the computational domain Γ. Using the prescribed value of the no-slip velocity and a fourth-order extrapolation formula, the values of the tangential velocity at the fictitious points are evaluated, thereby ensuring second-order accuracy for all spatial derivatives in equation 2.

The Poisson equation for the pressure subject to the homogeneous Neumann boundary condition is solved efficiently at each time step by employing a Correction-Scheme (CS) multigrid method which is appropriate for linear problems. At each level, the equations are relaxed by the well-known point red-black Gauss-Seidel scheme. Note that if M denotes the total number of multigrid levels, then at some level k, the 'W' cycle relaxes the equations 2^{M-k} more often than in the 'V' cycle where $k = M$ denotes the finest grid. In the scheme adopted here, the governing equation for the interior of the computational domain is relaxed twice in the forward sweep and once in the backward sweep; thus the 'V' and 'W' cycle scheme here may be denoted by $V(2,1)$ and $W(2,1)$, respectively. Furthermore, following Brandt [2], it was found effective to relax the boundary cells twice for each sweep of the interior cells.

Next, consider the idle-processor problem which occurs at a critical coarse-grid level where the total number of cells is less than the number of processors. The simplest approach is to have each processor solve the same problem at and below the critical level. In this case, denoted here as the 'lumped' scheme, it may be shown that the ideal efficiency (i.e. discounting all communication costs) is such that

$$(a)\ n \gg p \quad \eta_{ideal}^{-1} \sim 1 + O(\frac{1}{n}), \quad (b)\ n, p \gg 1 \quad \eta_{ideal}^{-1} \sim 1 + O(\frac{p}{n}), \tag{6}$$

where $n = N/p$. It is thus hoped that this scheme will be more effective than one which attempts to distribute the reduced extent of parallelism at the coarsest grids thus incurring relatively large latency cost due to frequent transmission of small messages.

3. IMPLICIT SCHEME

A spatial and temporal second-order accurate upwind-downwind discretization scheme [3] is applied to the computation of the unsteady incompressible Navier-Stokes equations on a staggered grid. A temporal discretization of the momentum equations at the mid-time plane, i.e. at $t + \Delta t/2$, yields

$$\frac{u^{n+1} - u^n}{\Delta t} = u\overline{\frac{\partial u}{\partial x}} + v\overline{\frac{\partial u}{\partial y}} - \frac{\partial p}{\partial x}^{n+\frac{1}{2}} + \frac{1}{Re}\left[\overline{\frac{\partial^2 u}{\partial x^2}} + \overline{\frac{\partial^2 u}{\partial y^2}}\right], \tag{7}$$

$$\frac{v^{n+1} - v^n}{\Delta t} = u\overline{\frac{\partial v}{\partial x}} + v\overline{\frac{\partial v}{\partial y}} - \frac{\partial p}{\partial y}^{n+\frac{1}{2}} + \frac{1}{Re}\left[\overline{\frac{\partial^2 v}{\partial x^2}} + \overline{\frac{\partial^2 v}{\partial y^2}}\right], \tag{8}$$

where the overbar denotes the evaluation of the quantities at the mid-time plane, and the superscripts n, $n+1$ and $n+\frac{1}{2}$ indicate the values of the associated variables at times t, $t+\Delta t$ and $t+\Delta t/2$, respectively. The resulting difference equations are given by [3]

$$\mathbf{M}_{+}\mathbf{q} = \mathbf{F} = \mathbf{M}_{-}\mathbf{q}^{*} + \mathbf{G}, \tag{9}$$

where

$$\mathbf{q} = \begin{bmatrix} u \\ v \\ p \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} G_u \\ G_v \\ G_p \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} F_u \\ F_v \\ F_p \end{bmatrix},$$

$$\mathbf{M}_{+} = \begin{bmatrix} M_{+}^{u} & 0 & D^{u} \\ 0 & M_{+}^{v} & D^{v} \\ D^{u} & D^{v} & 0 \end{bmatrix}, \quad \mathbf{M}_{-} = \begin{bmatrix} M_{-}^{u} & 0 & 0 \\ 0 & M_{-}^{v} & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The boundary conditions may also be written compactly as

$$\mathbf{\Lambda q} = \mathbf{\Phi}, \tag{10}$$

where $\mathbf{\Lambda}$ is the boundary operator and $\mathbf{\Phi}$ is specified [3]. In a typical solution procedure, equations 9 and 10 are relaxed alternately until convergence. For the implicit method, only the 'V' cycle is considered and the FAS multigrid algorithm, appropriate for nonlinear problems, is applied [2]. The equations are relaxed in a fully-coupled mode at each multigrid level by the PAR-SCGS algorithm [3], a parallel version of the Symmetrical-Coupled Gauss-Seidel (SCGS) algorithm [4], appropriate for distributed-memory machines using message passing. In this scheme, the four velocities at the faces of each cell and the pressure defined at the cell center are updated simultaneously in an iterative process that traverses all the cells in the computational domain.

In the implicit calculations, an alternative approach to that adopted in the explicit method is used for multigrid levels coarser than the critical level below which the number of cells is less than the number of processors [3]. Below the critical level, a group of four processors in a 2×2 grid coalesce their data and solve the same problem; in effect only $1/4th$ of the processors are active. The 'adjacent' neighbors are now a stride of 2 away in each of the coordinate directions. Similarly, at the next coarsening, a group of 16 processors solve the same problem and so on. For this algorithm, it may be shown that [3] that

$$(a)\ n \gg p \quad \eta_{ideal}^{-1} \sim 1 + O(\frac{1}{n}), \quad (b)\ n, p \gg 1 \quad \eta_{ideal}^{-1} \sim 1 + O(\frac{\log p}{n}). \tag{11}$$

Thus, the degradation in the ideal parallel efficiency is reduced to an acceptable level in comparison with the previous method for the case where both n and p are large.

4. RESULTS AND DISCUSSIONS

Table 1 shows the residual tolerance error that is specified for various fine grids. It is appropriate to choose the tolerance error to be of the same order of magnitude as the

Table 1
Residual tolerance error for various grid sizes

	Global Grid			
	32×32	64×64	128×128	256×256
Tol	1.0×10^{-3}	2.5×10^{-4}	6.0×10^{-5}	1.5×10^{-5}

discretization error which for a second-order accurate scheme is of the form Kh^2 where h denotes the mesh spacing and $K \sim O(1)$. Choosing $K = 1$, the residual tolerance error is then approximately set to h^{-2}. Convergence is deemed to have occurred at each time step when the residuals of all the equations are less than the residual tolerance. In the context of multigrid methods, it is convenient to quantify the computational effort in terms of work units (WU) [5]; a work unit is the computational effort required to relax the equations at the finest grid. It may be shown that for a two-dimensional computational domain, the number of WU's required for one cycle with $M > 1$ levels of multigrids is given by

$$WU = \frac{1}{1 - 2^{\mu-3}} \left[1 - 2^{(\mu-3)(M-1)}\right] (\nu_1 + \nu_2) + \nu_1 2^{(\mu-3)(M-1)-(\mu-1)}, \tag{12}$$

where $\mu = 1, 2$ for 'V' and 'W' cycles, respectively, and ν_1, ν_2 denote the number of iterations in the forward and backward sweeps, respectively. Note that in this study it is assumed that $\nu_1 = 2$ and $\nu_2 = 1$.

The results discussed here are for the classic test case of flow in a square cavity in which the top wall is set into motion impulsively at unit speed. Consider the explicit calculations first where $Re = 10^4$ and $\Delta t = 10^{-4}$ have been chosen. Table 2 shows the number of WU's required to obtain a converged solution at each time step as a function of the number of multigrid levels and resolution of the finest grid; the results are obtained by averaging the WU's over the first 10 time steps. It may be noted that the WU's decrease more rapidly for the 'W' cycle as the multigrid levels increase beyond one as compared to the 'V' cycle. However, beyond a certain level, indicated by an asterisk, the convergence rate of the 'W' cycle plateaus and no benefit is accrued in increasing the number of levels. On the other hand, the WU's decrease monotonically as the number of levels is increased for calculations with the 'V' cycle. This trend indicates that for comparable convergence rates, the number of multigrid levels required in the 'W' cycle is typically less than that required in the 'V' cycle; in the context of parallel computation, this is significant because inefficient calculations at the coarsest levels may be avoided in a 'W' cycle. All subsequent results reported here for 'W' cyles use the number of levels indicated by an asterisk in Table 2, but, for 'V' cyles, the full complement of available multigrid levels is employed.

For the 'lumped' scheme used here for the explicit calculations, the parallel efficiency is obtained from

$$\eta_{par} = t_{tot}^{-1} \left[t_{comp,dist} + \frac{t_{lumped}}{p} \right], \quad t_{tot} = t_{comp,dist} + t_{lumped} + t_{comm}, \tag{13}$$

where $t_{comp,dist}$ denotes the time for the calculations at the finer levels where the problem is distributed among all the processors, and t_{lumped} denotes the calculation time at the

Table 2
Work units for explicit calculations on a processor mesh of 8×4 ($Re = 10^4, \Delta t = 10^{-4}$).

Levels	Global Grid							
	32×32		64×64		128×128		256×256	
	V	W	V	W	V	W	V	W
1	110	110	-	-	-	-	-	-
2	39	39	259	259	-	-	-	-
3	13	10	71	45	749	471	-	-
4	6	*7	21	15	206	72	1753	617
5	5	7	10	*13	56	*19	456	88
6	-	-	9	13	19	19	118	*25
7	-	-	-	-	13	19	31	25
8	-	-	-	-	-	-	21	25

coarser levels where all the processors solve the same problem; these quantities do not include any communication time. Rather this time is t_{comm} which denotes the overall overhead in communication. The parallel efficiency for both the 'V' and 'W' cycles obtained from equation 13 is shown in figure 1 and compared with the efficiency for calculations performed only on the finest grid. It may be noted that the degradation in the efficiency of the 'W' cycle calculations is worse than that of the 'V' cycle calculations. Thus for the specific case considered here, the simpler 'V' cycle calculations are more effective; on the other hand, as indicated by the results in Table 2, if the size of the coarsest grid is sufficiently large, the 'W' cycle is likely to be the appropriate choice [6].

Next, consider the implicit calculation which employs the FAS multigrid algorithm with a 'V' cycle. Once again, the cavity flow test case is considered with $Re = 10^4$ and $\Delta t = 10^{-2}$, and the WU's shown in Table 3 are averaged over the first 10 time steps. The residual tolerance errors are as indicated in Table 1. A substantial reduction in computational effort may be noted as the number of grid levels is increased which becomes more pronounced as the number of mesh points in the finest grid is increased.

Table 3
Work units for implicit calculations on a processor mesh of 8×4 ($Re = 10^4, \Delta t = 10^{-4}$).

Levels	Global Grid			
	32×32	64×64	128×128	256×256
1	20	-	-	-
2	8	99	-	-
3	5	29	391	-
4	5	14	96	-
5	-	13	24	244
6	-	-	18	58
7	-	-	-	28

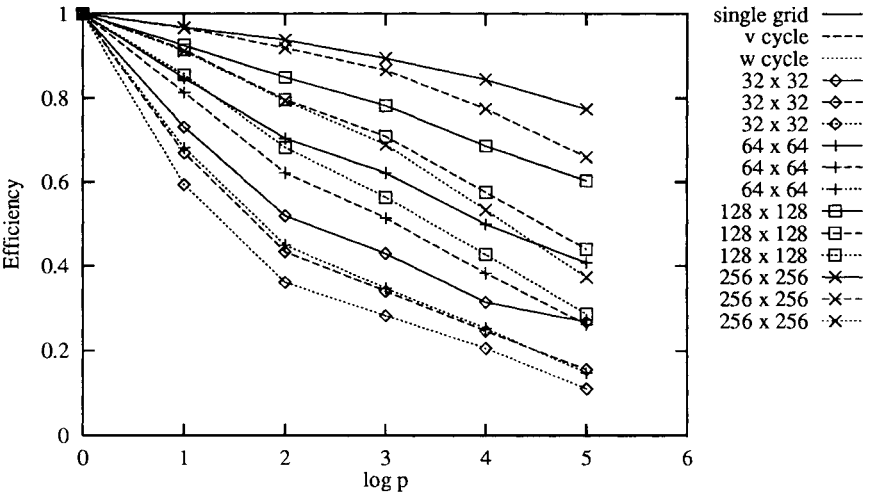


Figure 1. Variation of parallel efficiency with number of processors for single-grid and multigrid calculations for the explicit scheme.

The parallel efficiency for the implicit calculations, which uses the 'semi-distributive' scheme at the coarsest levels, is obtained from

$$\eta_{par} = t_{tot}^{-1}\left[t_{comp,dist} + \sum_{k=1}^{M_s} \frac{(t_{comp,semi})_k}{2^{2(M_s-k-1)}}\right], \quad t_{tot} = t_{comp,dist} + \sum_{k=1}^{M_s}(t_{comp,semi})_k + t_{comm}, \quad (14)$$

where M_s is the number of 'semi-distributive' levels and $(t_{comp,semi})_k$ is the computational time at the kth 'semi-distributive' level; $k = M_s$ is the finest 'semi-distributive' level. Figure 2 shows the parallel efficiency of the implicit calculations where the full complement of available multigrid levels is employed. The parallel multigrid efficiency is also compared to the efficiency of the calculations at the finest grid and it may be noted that the expected degradation in efficiency of the multigrid calculations is small for relatively large problems in comparison to calculations at the finest grid.

5. CONCLUSIONS

The parallel multigrid time-accurate calculation of the unsteady incompressible Navier-Stokes equations has been considered using both explicit and implicit methods. For both solution methods, it is clearly demonstrated that the computational effort required to obtain a converged solution at each time step reduces dramatically with increasing number of multigrid levels. Thus, although the parallel efficiency of the multigrid calculations is inferior to that possible for calculations only on the finest grid, the considerably superior

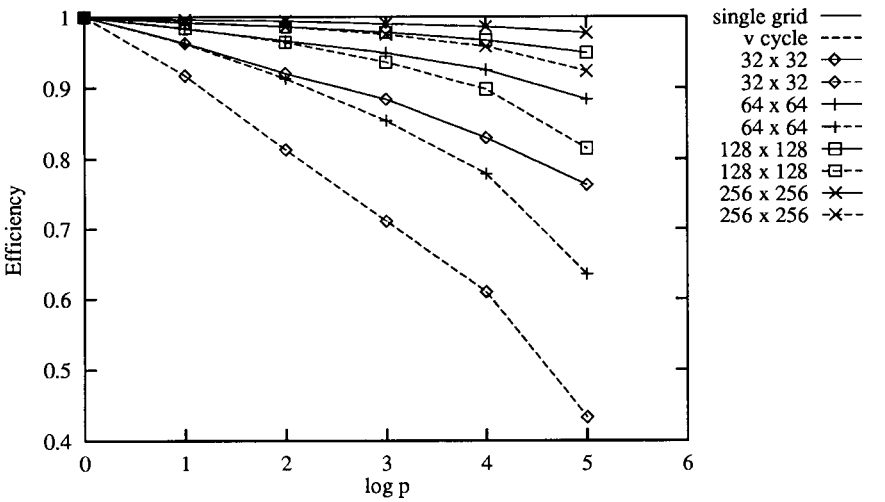


Figure 2. Variation of parallel efficiency with number of processors for single-grid and multigrid calculations for the implicit scheme.

convergence rates possible with the former dominates the degradation in parallel efficiency.

REFERENCES

1. R. Peyret and T. D. Taylor. *Computational Methods for Fluid Flow.* Springer-Verlag, 1983.
2. A. Brandt. *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics.* Number 85 in GMD-Studien. Gesellschaft fur Mathematik und Datenverarbeitung MBH, Bonn, 1984.
3. A. T. Degani and G. C. Fox. Parallel Computation of the Unsteady Incompressible Navier-Stokes Equations using Multigrids. In *12th AIAA CFD Conference*, AIAA Paper 95-1696, San Diego, CA, June 19-22, 1995.
4. S. P. Vanka. Block-Implicit Multigrid Solution of Navier-Stokes Equations in Primitive Variables. *Journal of Computational Physics*, 65:138–158, 1986.
5. A. Brandt. Multi-level Adaptive Solutions to Boundary-Value Problems. *Mathematics of Computation*, 31(138):333–390, 1977.
6. P. I. Crumpton and M. B. Giles. Parallel Unstructured Multigrid using OPlus. In *Parallel CFD 95*, Pasadena, CA, June 26-28, 1995.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Multigrid aircraft computations using the OPlus parallel library

Paul I. Crumpton and Michael B. Giles [a] *

[a]Oxford University Computing Laboratory, Parks Road, Oxford, OX1 3QD, UK

This paper presents the OPlus library which is a flexible library for distributed memory parallel computations on unstructured grids through the straightforward insertion of simple subroutine calls. It is based on an underlying abstraction involving sets, pointers (or mappings) between sets, and operations performed on sets. The key restriction enabling parallelisation is that operations on a particular set can be performed in any order.

The set partitioning, computation of halo regions, and the exchange of halo data as required is performed automatically by the OPlus library after the user specifies the sets and pointers. A single source OPlus application code can be compiled for execution on either a parallel or a sequential machine, greatly easing maintainability of the code.

The capabilities of the library are demonstrated by its use within a program for the calculation of inviscid flow over a complete aircraft using multigrid on a sequence of independent tetrahedral grids. Good computational efficiency is achieved on an 8-processor IBM SP1 and a 4-processor Silicon Graphics Power Challenge.

1. INTRODUCTION

Algorithms for unstructured grids are becoming increasingly popular, especially within the CFD community where the geometrical flexibility of unstructured grids enables whole aircraft to be modelled. The resulting calculations are often huge and so there is a need to fully exploit modern distributed memory parallel computers. Writing an individual, machine-specific parallel program can be time consuming, expensive and difficult to maintain. Therefore there is a need for tools to simplify the task and generate very efficient parallel implementations. This paper describes the development of OPlus (Oxford Parallel library for unstructured solvers), a FORTRAN subroutine library which enables the parallelisation of a large class of applications using unstructured grids, removing the parallelisation burden from the application programmer as far as possible [1].

In the design of the library emphasis was placed on the following aspects:

generality: OPlus uses general data structures. In a CFD application, for example, it allows cell, edge, face and other data structures, The cells could also be of any type, such as tetrahedra, prisms or hexahedra.

performance: Messages are sent only when data has been modified and are concatenated to reduce latency. Also, local renumbering is used to improve the cache performance on RISC processors.

*Research supported by Rolls-Royce plc, EPSRC and DTI

single source: A single source code can be executed either sequentially (without any message-passing of other parallel library) or in parallel, with identical treatment of disk and terminal i/o. This greatly simplifies development and maintenance of parallel codes.

This paper will first describe the concepts behind the OPlus framework, and then various aspects of the implementation. Finally some results are presented for an application code modelling the inviscid flow over an aircraft using multiple tetrahedral meshes. A companion paper discusses the use of the distributed visualisation software pV3 which was a vital tool in this work [5].

The PARTI library developed by Das *et al* [7,6] has similar objectives in dealing with parallel computations on generic sets. There are a number of detailed differences between PARTI and OPlus but the principal difference is that with OPlus the programmer is not aware of the message-passing required for the parallel execution. This greatly simplifies the programmer's task. PARTI has the same long-term objective but the aim is to achieve it through the incorporation of PARTI within an automatic parallelising compiler. At present, the programmer must still explicitly specify the message-passing to be performed.

2. OPlus LIBRARY

2.1. Top level concept

The concept behind the OPlus framework is that unstructured grid applications have three key components.

sets Examples of sets are nodes, edges, triangular faces, quadrilateral faces, cells of a variety of types, far-field boundary nodes, wall boundary faces, etc. Data is associated with these sets, for example the grid coordinates at nodes, the volumes of cells and the normals on faces.

pointers The connectivity of the computational problem is expressed by pointers from elements of one set to another. For example, cell to node connectivity could define tetrahedra, and face to node connectivity would define the corresponding faces.

operations over sets All of the numerically-intensive parts of unstructured applications can be described as operations over sets. For example, looping over the set of cells using cell–node pointers and nodal data to calculate a residual and scatter an update to the nodes, or looping over the nonzeros of a sparse matrix accumulating a matrix-vector product.

The OPlus framework makes the important restriction that an operation over a set can be applied in *any* order without affecting the final result. Consequently, the OPlus routines can choose an ordering to achieve maximum parallel efficiency. This restriction prevents the use of OPlus for certain numerical algorithms such as Gauss–Seidel iteration or globally implicit approximate factorisation time-marching methods. However, most numerical algorithms on unstructured grids in current use in CFD, and many other application areas, satisfy this restriction. Specific examples include explicit time-marching methods, multigrid calculations using explicit smoothing operators and conjugate gradient methods using local preconditioning.

Another current restriction is that the sets and pointers are declared at the start of the program execution and must then remain unaltered throughout the computation. Therefore, dynamic grid refinement cannot be treated at present. This is an area for future development.

2.2. Parallelisation approach

The implementation uses a standard data-parallel approach in which the computational domain is partitioned into a number of regions and each partition is treated by a separate process, usually on a separate processor.

The communication requirements between partitions arise because of the pointer connectivity at the boundaries between partitions. An illustrative example is matrix-vector multiplication for a symmetric sparse matrix:

$$y_i = \sum_j A_{ij} x_j$$

If the matrix A is symmetric, then defining an edge k to correspond to nodes i_k, j_k for which $A_{ij} \neq 0$, the product can be evaluated by the following algorithm:

For all nodes i, $y_i := 0$

For all edges k, $y_{i_k} := y_{i_k} + A_k x_{j_k}$
$y_{j_k} := y_{j_k} + A_k x_{i_k}$

Expressed in FORTRAN this algorithm becomes

```
      DO I = 1, NNODES
        Y(I) = 0.0
      ENDDO
C
      DO K = 1, NEDGES
        I    = P(1,K)
        J    = P(2,K)
        Y(I) = Y(I) + A(K)*X(J)
        Y(J) = Y(J) + A(K)*X(I)
      ENDDO
```

The integer array P is the pointer table defining the edge to node connectivity. Note that operations on edges can be performed in any order and the final result will remain the same, so this example satisfies the restriction required by the OPlus framework.

In the data parallel approach the edges and nodes are partitioned so that each individual edge or node *belongs* to only one partition. There is no difficulty in performing the edge operations when the edge and its two nodes belong to the same partition. When more than one partition is involved the approach used is to perform the edge operation on

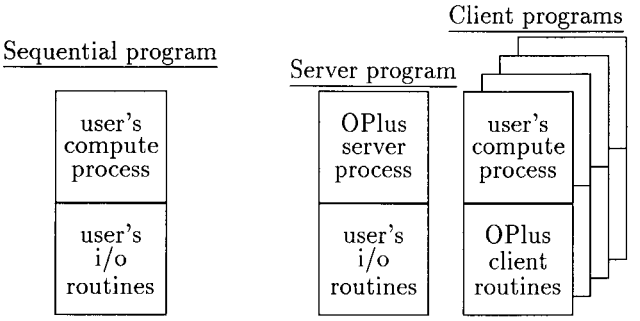


Figure 1. Sequential and parallel versions of user's program

each partition whose owned data is affected by the operation. In this case that means performing the calculation for each partition owning one of the two nodes. To carry out this operation, temporary copies must be obtained of the unowned data from the other node and/or the edge belonging to a different partition; this is commonly referred to as *halo data.*

2.3. Software architecture and communications

A key design goal for the OPlus framework was to allow users to write a *single source code* which will execute either sequentially or in parallel depending only on the library to which it is linked. Moreover, the sequential and parallel execution should result in identical disk and terminal i/o. To achieve this it was necessary to adopt the program structure shown in Figure 1 in which all disk and terminal i/o is handled via subroutines with a specified interface.

For sequential execution the user's main program is linked to user–written subroutines which handle all i/o. This enables the user to develop, debug and maintain their sequential code without any parallel message passing libraries.

For parallel execution the OPlus framework creates server and client programs from the user's single source. The server program is formed by linking the OPlus server process to the user's i/o routines, while the client program is created by linking the user's compute process to the OPlus client routines. When the client routines collectively need to input an array of data from disk, a request is passed to the server process; it reads the data from disk and passes to each client its piece of the array corresponding to its partition of the overall problem.

The communication between the client and the server is performed using PVM 3.3 to allow the server to be on a different type of machine from the clients. For the critical client-client communication in the main parallel computation phase, BSP FORTRAN is employed. This is a FORTRAN library with strong similarities to the `shmem_put` and `shmem_get` directives for communication on the CRAY–T3D. It has been implemented on a wide range of machines using the most efficient communication method in each case, e.g. MPL on the IBM SP1/SP2 and shared-memory pages on the Silicon Graphics Power Challenge [8].

2.4. Initialisation phase

At the beginning of the application program the user declares the sets and pointers to be used in the application. At the beginning of the parallel execution these are then used in the following key steps:

partitioning All sets are partitioned. Simple recursive inertial bisection is used to initially partition one or more sets. The other sets are partitioned consistently using the connectivity information contained in the pointers; see [5] for examples of inherited partitioning.

construction of import/export lists The initialisation phase constructs, for each partition, lists of the set members which may need to be imported during the main execution phase. Correspondingly, each partition also has export lists of the owned data which may need to be imported by other partitions.

local renumbering Each partition should only need to allocate sufficient memory to store the small fraction of each set which it either owns or imports. To enable this, it is necessary to locally renumber the set members. This local renumbering of each set forces a consistent renumbering of all of the pointer information. The local-global mapping is also maintained for i/o purposes.

It is important to note again that all of the above phases are performed automatically by the OPlus library, not the application code. In all applications performed to date, the CPU time taken for these initialisation phases has been significantly less than the time required for the disk i/o, and so is considered to be negligible.

2.5. Computation phase

The heart of a parallel OPlus application is a DO-loop carrying out in parallel operations on a distributed partitioned set. Continuing with the example of the sparse matrix-vector product, using the OPlus library the FORTRAN code for the main edge loop is:

```
  DO WHILE(OP_PAR_LOOP(EDGES,K1,K2))
    CALL OP_ACCESS('read'  ,X,1,NODES,P,2,1,1,1,2)
    CALL OP_ACCESS('update',Y,1,NODES,P,2,1,1,1,2)
    CALL OP_ACCESS('read'  ,A,1,EDGES,0,0,1,1,0,0)
C
    DO K = K1, K2
      I    = P(1,K)
      J    = P(2,K)
      Y(I) = Y(I) + A(K)*X(J)
      Y(J) = Y(J) + A(K)*X(I)
    ENDDO
  END WHILE
```

The purpose of the OP_ACCESS calls is to inform the library which distributed arrays are being used within the DO-loop, which sets they are associated with, which pointers are being used to address those sets and the type of operation (read, write or update) being undertaken with the data. This is the information needed by the library to decide which data must be imported from neighbouring partitions. Full details of the arguments of OP_ACCESS and the other OPlus routines are available [2].

The logical function OP_PAR_LOOP controls the number of times execution passes though the interior of the DO WHILE loop. The first argument declares that the operations are to be performed over the set of edges, and the second and third arguments set by the function are the start and finish indices of the inner loop. For sequential execution, there is just one pass through the DO WHILE loop with K1 and K2 set to 1 and NEDGES respectively. For parallel execution there are a number of preliminary passes through the DO WHILE loop during which K1 is set to a higher value than K2 so the inner DO loop is skipped; these passes process the information in the OP_ACCESS calls and export the necessary halo data to neighbouring partitions. Next comes a single pass through the DO loop performing those calculations which do not depend on halo data. There are then a number of passes which receive the incoming imported data from neighbouring partitions but perform no calculations, and finally there is an execution pass which performs the computations that do depend on the halo data. In this way it is possible to overlap interior computations with the exchange of halo data, but so far this overlapping has not yielded significant benefits on any of the machines tested.

A point to emphasise is that the lines of FORTRAN which form the contents of the inner DO loop have not changed from the original sequential code. In this trivial example the number of OPlus subroutine calls which have been added is comparable to the number of lines of application code, but in a real application there could be a hundred lines or more of FORTRAN inside the DO loop and it is crucial that this does not have to be changed.

3. MULTIGRID AIRCRAFT COMPUTATION

The utility and efficiency of the OPlus library is illustrated here by its use for the computation of the steady inviscid flow around a complex aircraft geometry. The CFD algorithm uses a multigrid procedure based on a Lax-Wendroff solution algorithm [3,4]. In this application five tetrahedral grids are used. The number of cells varies from 750k on the finest grid to 28k on the coarsest. The surface triangulation of one of the grids is shown in Figure 2 together with the final surface pressure contours. Previous research [3] showed that a W-cycle multigrid iteration is twice as fast as a V-cycle iteration, and so a W-cycle iteration is used in this work. However, this presents a great challenge for parallel efficiency because of the very large number of iterations performed on the coarsest grids which have relatively more communication and redundant computation.

On each of the five grids there are four sets, tetrahedra, nodes, boundary faces and boundary nodes. For each grid there are pointers from the tetrahedra, boundary faces and boundary nodes to the regular nodes. There are also pointers between grid levels, giving for each node the four nodes of the enclosing tetrahedra on the finer and coarser grids. These are required for the transfer and interpolation operations within the multigrid

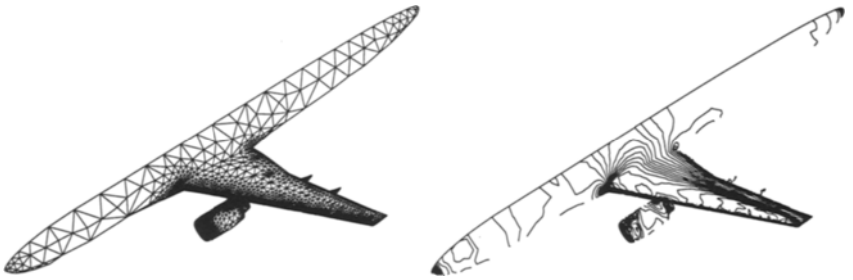


Figure 2. One of the grids used for the aircraft computation and computed contours of surface pressure

procedure.

Calculations were performed on an 8 node distributed memory IBM SP1 and a 4 processor shared memory SGI Power Challenge. The elapsed times, in seconds per multigrid cycle, and the corresponding speedup over the single processor performance, S, is given in the following table.

		IBM SP1		SGI PC	
proc	$S_{\max}$	time	S	time	S
1	1.0	1006	1.0	419	1.0
2	1.9	556	1.8	216	1.9
3	2.7	384	2.6	149	2.8
4	3.5	310	3.2	116	3.6
5	4.1	270	3.7	—	—
6	4.8	234	4.3	—	—
7	5.4	211	4.8	—	—
8	6.1	190	5.3	—	—

The maximum achieveable speed-up, $S_{\max}$, is defined as the ratio of the total sequential work to the maximum work performed on any of the OPlus clients,

$$S_{\max} = \frac{\text{sequential work}}{\text{max slave work}}.$$

This is the speedup which would be achieved in the complete absence of communication costs. There are two reasons why it does not increase linearly with the number of processors. One is that it is difficult to achieve load-balancing on the coarser grid levels; the grid nodes may be load-balanced but the boundary faces may not be. The other is that redundant computations are performed at the interfaces between partitions; the proportion of these becomes larger on the coarser grids. Nevertheless, very good parallel execution speed up has been achieved for a highly complex application which is in many ways a worst case in that it employs the W-cycle multigrid iteration which is known to be a challenge to efficient parallel execution. Note that the factor of two saved by the use of W-cycle multigrid compared to V-cycle multigrid is still much greater than any loss of

parallel efficiency associated with the increased number of iterations on the coarse grid levels.

4. CONCLUSIONS

A flexible and general library has been developed to parallelise a large class of unstructured grid applications. The programmer specifies the sets and pointers to be used in the application and the library determines an appropriate partitioning for data-parallel execution. The transfer of halo data is performed automatically by the library given the programmer's specification of the data being used in operations performed on the members of the distributed sets. The resulting single source code will execute on a sequential machine without the need for any parallel libraries, or in parallel on a MIMD architecture.

The use of the OPlus library has been demonstrated for a multigrid computation of the inviscid compressible flow over a complete aircraft configuration. For this realistic industrial application good parallel efficiency has been achieved with very little effort from the application programmer.

REFERENCES

1. D.A. Burgess, P.I. Crumpton, and M.B. Giles. A parallel framework for unstructured grid solvers. In S. Wagner, E.H. Hirschel, J. Périaux, and R. Piva, editors, *Computational Fluid Dynamics '94. Proceedings of the Second European Computational Fluid Dynamics Conference 5-8 September 1994 Stuttgart, Germany*, pages 391–396. John Wiley & Sons, 1994.
2. P. Crumpton and M. Giles. OPlus programmer's guide, rev. 1.0. 1993.
3. P. Crumpton and M.B. Giles. Aircraft computations using multigrid and an unstructured parallel library. AIAA Paper 95-0210, 1995.
4. P.I. Crumpton and M.B. Giles. Implicit time accurate solutions on unstructured dynamic grids. AIAA Paper 95-1671, 1995.
5. P.I. Crumpton and R. Haimes. Parallel visualisation of unstructured grids. In S. Taylor, A. Ecer, J. Periaux, and N. Satofuka, editors, *Proceedings of Parallel CFD'95, Pasadena, CA, USA 26–29 June*, 1995.
6. R. Das, D.J. Mavriplis, J. Saltz, S. Gupta, and R. Ponnusamy. Design and implementation of a parallel unstructured Euler solver using software primitives. *AIAA Journal*, 32(3):489–496, 1994.
7. R. Das, J. Saltz, and H. Berryman. A manual for PARTI runtime primitives, revision 1. Technical report, ICASE, NASA Langley Research Centre, Hampton, USA, 1993.
8. R. Miller. A library for bulk synchronous parallel programming. In *Proceedings of the BCS Parallel Processing Special Interest Group Workshop on General Purpose Parallel Computing*, Dec 1993. http://www.comlab.ox.ac.uk/oucl/oxpara/ppsg.html.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A Parallel Multigrid Method for the Prediction of Turbulent Flows with Reynolds Stress Closure

B. Basara[a], F. Durst[b] and M. Schäfer[b]

[a]Department of Civil Engineering, City University,
London EC1V 0HB, United Kingdom

[b]Department of Fluid Mechanics, University of Erlangen,
Cauerstr. 4, D-91058 Erlangen, Germany

In this paper a parallel nonlinear multigrid method for the computation of turbulent flows in complex geometries with second-order closure turbulence modeling is presented. The numerical solution procedure is based on a collocated blockstructured finite volume discretization, a full approximation scheme, a pressure-correction procedure for smoothing and coupling of variables, and a grid partitioning technique for parallelization. Considering several test cases the capabilities of the considered approach are investigated with respect to numerical and parallel efficiency. The results also include comparisons with corresponding computations with a k-ϵ model.

1. INTRODUCTION

Practically relevant flows in more than 95 per cent of the cases are turbulent, and thus, since for higher Reynolds numbers *large eddy simulations* or *direct numerical simulations* are out of reach, one is faced with the problem of adequate turbulence modeling. It is well known that models based on the Boussinesq approximation for the Reynolds stresses (as the k-ϵ or k-ω models), which are frequently in use in practice, are unsatisfactory for a wide range of applications (e.g. Wilcox [11]). Examples, where these models fail are flows with boundary-layer separation, buoyancy-driven flows, duct flows with secondary motions, flows over curved surfaces, flows in rotating and stratified fluids, or flows with sudden changes in mean strain rate. Second order closure models, which are based on exact differential equations for the Reynolds stresses, allow for a natural capturing of such effects. However, for this higher modeling accuracy one has to pay with a higher complexity of the equations and an increased computational difficulty to get a numerical solution. Thus, when dealing with such approaches it is especially important to consider advanced solution algorithms as well as high performance computer architectures.

In the present work a numerical solution method is presented, which combines advanced second order moment closure turbulence modeling, numerically efficient multigrid techniques, and parallel computing. To the authors knowledge this is the first time such an approach is employed for the computation of turbulent flows. A sequential multigrid method, which however differs from the present one in several aspects, applied for the full

Reynolds-stress-transport closure is presented in the work of Lien and Leschziner [6].

In the next sections we briefly discuss the employed second order moment closure model, which follows the recent approach of Speziale, Sarkar and Gatski [9], the employed numerical procedure based on a finite volume discretization for non-orthogonal blockstructured boundary-fitted grids with a collocated arrangement of variables, and a parallel multigrid procedure with iterative pressure-correction smoothing for the efficient solution of the resulting coupled system of algebraic equations.

By considering two two-dimensional test cases the performance of the considered approach is investigated. The second order moment closure approach is also compared with corresponding computations using a standard k-ϵ model. These comparisons relate to convergence issues of the parallel multigrid method, parallel efficiency, and the overall efficiency of the solution procedure.

2. GOVERNING EQUATIONS

For steady incompressible fluid flows, to which we restrict ourselves in this paper, the Reynolds averaged Navier-Stokes equations expressing balance of mass and momentum are given by

$$\frac{\partial U_i}{\partial x_i} = 0, \tag{1}$$

$$U_j \frac{\partial U_i}{\partial x_j} = \frac{\partial}{\partial x_j}(\nu \frac{\partial U_i}{\partial x_j} - \overline{u_i u_j}) - \frac{1}{\rho}\frac{\partial p}{\partial x_i}, \tag{2}$$

where U_i are the mean-velocity vector components with respect to Cartesian coordinates x_i, p is the static pressure, ν is the kinematic viscosity, and $\overline{u_i u_j}$ are the Reynolds stresses. With a second order moment closure of the above system the Reynolds stresses are obtained from transport equations of the form

$$U_k \frac{\partial \overline{u_i u_j}}{\partial x_k} = \phi_{ij} + P_{ij} + d_{ij} + \epsilon_{ij}. \tag{3}$$

The terms on the right hand side of (3) correspond to the following different physical processes: redistribution ϕ_{ij}, production P_{ij}, diffusion d_{ij}, and dissipation ϵ_{ij}. In the present model these quantities are defined as follows:

$$\phi_{ij} = \overline{\frac{p'}{\rho}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)}, \tag{4}$$

$$P_{ij} = -(\overline{u_i u_k}\frac{\partial U_j}{\partial x_k} + \overline{u_j u_k}\frac{\partial U_i}{\partial x_k}), \tag{5}$$

$$d_{ij} = \frac{\partial}{\partial x_k}\left[\nu \frac{\partial \overline{u_i u_j}}{\partial x_k} + C_s \frac{k}{\epsilon}\overline{u_k u_l}\frac{\partial \overline{u_i u_j}}{\partial x_l}\right], \tag{6}$$

$$\epsilon_{ij} = -2\delta_{ij}\epsilon/3. \tag{7}$$

The major modeling aspects are: the diffusion of pressure fluctuations is neglected, the triple fluctuating velocity correlations are modeled by the gradient transport hypothesis of Daly and Harlow [2] for the dissipation of turbulence fluctuations, the dissipation of turbulent energy is assumed to be isotropic, and the fluctuating pressure-strain term in ϕ_{ij} is treated following the recent approach of Speziale, Sarkar and Gatski [9].

For the dissipation rate of turbulent kinetic energy the following equation is adopted (this differs from the one used in the standard k-ϵ model):

$$U_j \frac{\partial \epsilon}{\partial x_j} = \frac{\partial}{\partial x_j}(C_\epsilon \frac{k}{\epsilon} \overline{u_j u_l} \frac{\partial \epsilon}{\partial x_l}) - C_{\epsilon_1} \frac{\epsilon}{k} \overline{u_i u_j} \frac{\partial U_i}{\partial x_j} - C_{\epsilon_2} \frac{\epsilon^2}{k}. \tag{8}$$

Equations (1), (2), (3), and (8) form the system of partial differential equations which has to be solved for U_i, p, $\overline{u_i u_j}$, and ϵ. For the boundary conditions at non-permeable walls the standard wall function approach is employed (see e.g. Wilcox [11]). More details about the employed model, including also the proper values for the various constants, can be found in the work of Basara and Younis [1].

3. NUMERICAL METHOD

The numerical solution method is based on a fully conservative finite volume discretization on blockstructured non-orthogonal boundary-fitted grids with a non-staggered arrangement of variables (e.g. Durst *et al.* [3]), where the well known interpolation technique of Rhie and Chow [8] is employed to ensure the coupling of the solution. Second order discretization is used for all terms (central differences, linear interpolation) together with a defered correction approach for the convective fluxes.

The continuity equation is used to obtain a pressure-correction equation according to a variant of the SIMPLE algorithm of Patankar and Spalding [7]. The linearized equations for velocity components, pressure-correction, and turbulence quantities are assembled and relaxed sequentially, where as linear system solver the ILU approach of Stone [10] is employed. Outer iterations are performed to take into account the non-linearity, the coupling of variables, and effects of grid non-orthogonality, which are treated explicitly in all equations, and under-relaxation is used to ensure the convergence of the iterative procedure.

For accelerating the rate of convergence a nonlinear multigrid method (full approximation scheme), in which the pressure-correction scheme acts as the smoother, is employed. V-cycles are used for the movement through the grid levels and nested iteration is employed for improving the initial guesses on the finer grid levels (full multigrid). The basic concepts of the multigrid technique are described in detail by Hortmann *et al.* [5]. There are some special aspects of the multigrid approach related to the second order Reynolds stress modeling (RSM): RSM on coarsest grid starts with a converged solution obtained from a computation with a k-ϵ model, near boundary values are extrapolated from interior grid points for the initial coarse to fine grid transfers (instead of the usual bilinear interpolation), the values of k obtained from restriction are kept fixed during the coarse grid pressure-correction iterations, and values of ϵ and $\overline{u_i u_i}$ are not corrected, if the corrected values would become negative.

The parallelization of the method is achieved by a grid partitioning technique based on the blockstructuring. For the determination of a suitable partitioning an automatic load balancing procedure is implemented. According to the number of available processors a mapping of the geometrical blockstructure to a parallel blockstructure is performed such that the resulting subdomains can be suitably assigned to the available processors with respect to a balanced utilization during the computation. For handling the coupling of the blockstructured grids auxiliary control volumes containing the corresponding boundary values of the neighbouring block are introduced along the block interfaces. The coupling of the blocks is ensured by communicating these boundary values during the iterative solution algorithm. A flow diagram of the parallel procedure is shown in Figure 1.

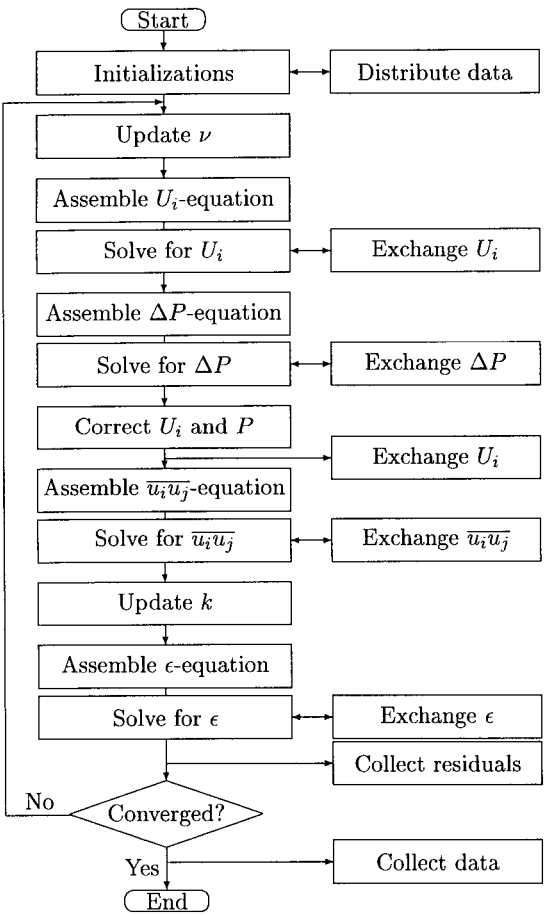


Figure 1. Flow chart of the parallel pressure-correction scheme.

It should be noted that the parallel multigrid method is implemented globally, i.e. with-

out being affected by the grid partitioning, such that a close coupling of the subdomains and, therefore, only a slight deterioration of the numerical efficiency compared to the corresponding sequential algorithm is ensured. More details of the overall parallel solution procedure are given in the paper by Durst and Schäfer [4].

4. NUMERICAL RESULTS

All numerical results given in the following were obtained on a Parsytec GC/PowerPlus parallel computer employing the communication facilities under the Parix operating system. In all cases the pressure-correction smoother was employed with relaxation factors 0.8 for velocity components, 0.2 for pressure, and 0.9 for all turbulence quantities.

As a first example for investigating the computational efficiency of the proposed numerical procedure a simple turbulent channel flow is considered. This test case allows to concentrate on the effects related to the parallel multigrid methodology together with the turbulence model without being effected from aspects of geometrical complexity. At the inflow a block velocity profile $(U, V) = (U_m, 0)$, to give a Reynolds number of $Re = 10^7$, is specified and the inlet values for the turbulence quantities are chosen as

$$\overline{u_1 u_1} = 1.1E_m,\ \overline{u_2 u_2} = 0.25E_m,\ \overline{u_3 u_3} = 0.65E_m,\ \overline{u_1 u_2} = 0,\ \epsilon = 10E_m^{3/2} \tag{9}$$

with $E_m = 0.01U_m^2$.

The problem was computed with the Reynolds stress model (RSM) given in Section 2 as well as, for comparison, with the standard k-ϵ model (e.g. Wilcox [11]). For both cases the multigrid procedure was used with (10,10,10)-V-cycles with a coarsest grid of 64 control volumes (CVs).

In Table 1 the computing times (seconds) and the numbers of fine grid iterations (in brackets) are given for different numbers of CVs and processors together with the corresponding single-grid results. The resulting acceleration factors (with respect to computing time) are also indicated.

Table 1
Comparison of single-grid and full multigrid computations with different numbers of CVs and processors for the channel flow with RSM and k-ϵ model. Computing times (seconds), numbers of fine grid iterations (in brackets) and corresponding acceleration factors.

	4,096 CV		16,384 CV	65,536 CV	
k-ϵ	P=1	P=4	P=4	P=4	P=16
Single-grid	264(248)	122(293)	1279(1045)	17501(3929)	7863(4955)
Full multigrid	114(61)	97(61)	257(81)	903(101)	699(101)
Acceleration	2.3	1.3	5.0	19.4	11.2
RSM					
Single-grid	476(260)	199(297)	2406(1133)	32607(4047)	12604(4913)
Full multigrid	275(89)	189(101)	709(161)	3397(243)	2066(261)
Acceleration	1.7	1.1	3.3	9.6	6.1

Several conclusions can be drawn from the results. In all cases the use of the multigrid method results in an acceleration of the computations. As usual, the acceleration factor increases with the number of CVs, and it decreases with the number of processors because of the higher communication effort due to the coarse grid computations. While still significant, at least for finer grids, the multigrid acceleration is lower as in comparable laminar cases (see e.g. Durst and Schäfer [4]). This fact seems to be related to the very steep gradients occuring along walls and the wall function approach. Independently of the numbers of CVs and processors the multigrid acceleration for the RSM computations is slightly lower as for the corresponding k-ϵ computations and, in general, the computational effort is is about 2-3 times higher for the RSM computations. The parallel efficiency is similar for the RSM and k-ϵ computations and the grid partitioning only slightly deteriorates the convergence of the multigrid method due to its global implementation (compare e.g. iteration numbers for 4,096 CVs with $P = 1$ and $P = 4$ or for 65,536 CVs with $P = 4$ and $P = 16$, where the iteration number is even uneffected for the k-ϵ computations). This behaviour as well as the parallel efficiency is comparable to laminar cases (see e.g. Durst and Schäfer [4]).

As a second example, involving a more complex non-orthogonal geometry, the flow around a circular cylinder in a channel is considered. Figure 2 shows the configuration together with the numerical grid, the partitioning for 16 processors, as well as the computed distribution of the Reynolds stress $-\rho\overline{u_2 u_2}$. Again, the Reynolds number is $Re = 10^7$ and the inflow conditions are chosen according to (9). The multigrid procedure is used with (20,20,20)-V-cycles with a coarsest grid of 256 CVs (in Figure 2 the grid with 4,096 CVs is shown).

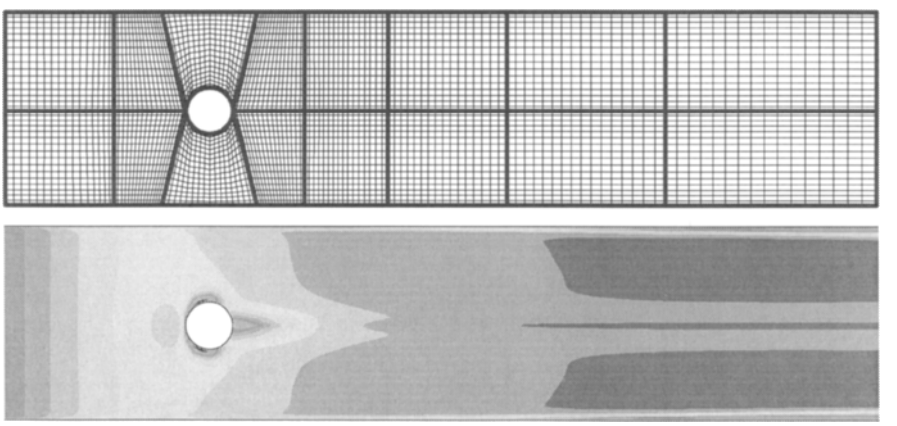


Figure 2. Geometry, numerical grid, partitioning for 16 processors, and computed distribution of Reynolds stress $-\rho\overline{u_2 u_2}$ for the flow around a circular cylinder.

In Table 2 the computing times (minutes) and the numbers of fine grid iterations (in brackets) are given for computations with the RSM and the k-ϵ model. The results are

given for the single-grid method with nested iteration and the full multigrid method together with the resulting acceleration factors for different numbers of CVs and processors, where in each case the number of CVs per processor is the same. It should be noted that without nested iteration for this problem no convergence can be obtained for the chosen set of under-relaxation factors. Of course, convergence can be enforced by using very small values of these factors for velocity components and turbulence quantities, but the rate of convergence would be very poor.

Table 2
Comparison of single-grid (with nested iteration) and full multigrid computations with different numbers of CVs and processors for the flow around a circular cylinder with RSM and k-ϵ model. Computing times (minutes), numbers of fine grid iterations (in brackets), and corresponding acceleration factors.

k-ϵ	16,384 CV, P=1	65,536 CV, P=4	262,144 CV, P=16
Single-grid (NI)	71(887)	186(2267)	510(5641)
Full multigrid	25(201)	36(241)	74(361)
Acceleration	2.8	5.2	6.9
RSM			
Single-grid (NI)	135(954)	373(2622)	1026(7002)
Full multigrid	56(281)	91(401)	223(847)
Acceleration	2.4	4.1	4.6

Comparing the results in Table 2 with that for the channel flowin Table 1 one can see that, in principle, the same conclusions as stated above for the latter are valid. A difference can be observed in the acceleration factors, which are lower for the cylinder flow. Of course, this mainly has to do with the better performance of the single-grid method due to the additionally employed nested iteration, but there is also an influence of the higher flow complexity. Looking at the results for 65,536 CVs with $P = 4$, which are indicated for both cases, one can see that for the cylinder flow the number of fine grid iterations for the full multigrid computation is significantly higher. The ratio is 1.7 for the RSM model and it is 2.4 for the k-ϵ model.

Another experience of the authors, which could be made during the test computations and is worth to be noted, is that, in general, the multigrid method stabilizes the RSM as well as the k-ϵ computations, which means that the method is more robust with respect to the choice of under-relaxation factors and grid distortions.

5. CONCLUSIONS

Simulations of turbulent flows with second-order moment closure modeling are very ambitious with respect to computer resources. The results in this paper indicate that with an efficient exploitation of computing power provided by modern parallel computers in connection with advanced numerical techniques like multigrid methods a significant acceleration of such computations, especially for fine grids, can be achieved. Thus, employing such techniques, gives the possibility to treat such problems with high accuracy

(fine grids) within a reasonable amount of computing time, and also allows for a much wider application of such improved turbulence modeling approaches for the investigation of complicated practical turbulent flow problems. It can easily be foreseen, that due to such improvements computations with advanced second-order closure models, for which is a strong need in many kinds of flow situations, will significantly increase in practice.

Acknowledgements

The authors would like to thank Prof. N. Stošić for his support and many helpful discussions. The financial support by the *Bayerische Forschungsstiftung* in the *Bavarian Consortium of High-Performance Scientific Computing (FORTWIHR)* and the *Deutsche Forschungsgemeinschaft* in the priority research programme *Flow Simulation with High-Performance Computers* is gratefully acknowledged.

REFERENCES

1. B. Basara and B.A. Younis. Assessment of the SSG Pressure-Strain Model in Two-Dimensional Turbulent Separated Flows. *Applied Scientific Research*, 1995. To appear.
2. B.J. Daly and F.H. Harlow. Transport equations in turbulence. *Physics of Fluids*, 13:2634–2649, 1970.
3. F. Durst, M. Perić, M. Schäfer, and E. Schreck. Parallelization of Efficient Numerical Methods for Flows in Complex Geometries. In *Flow Simulation with High-Performance Computers I*, volume 38 of *Notes on Numerical Fluid Mechanics*, pages 79–92. Vieweg Verlag, 1993.
4. F. Durst and M. Schäfer. A Parallel Blockstructured Multigrid Method for the Prediction of Incompressible Flows. *Int. J. for Num. Meth. in Fluids*, 1996. To appear.
5. M. Hortmann, M. Perić, and G. Scheuerer. Finite volume multigrid prediction of laminar natural convection: Benchmark solutions. *Int. J. Num. Meth. in Fluids*, 11:189–207, 1990.
6. F.S. Lien and M.A. Leschziner. Multigrid Acceleration for Recirculating Laminar and Turbulent Flows Computed with a Non-Orthogonal, Collocated Finte-Volume Scheme. *Computer Methods in Applied Mechanics and Engineering*, 118:351–371, 1994.
7. S.V. Patankar and D.B. Spalding. A calculation procedure for heat, mass and momentum transfer in three dimesional parabolic flows. *Int. J. Heat Mass Transfer*, 15:1787–1806, 1972.
8. C.M. Rhie and W.L. Chow. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal*, 21:1525–1532, 1983.
9. C.G. Speziale, S. Sarkar, and T.B. Gatski. Modelling the pressure-strain correlation of turbulence, an invariant dynamical systems approach. *Journal of Fluid Mechanics*, 227:245–272, 1990.
10. H. Stone. Iterative solution of implicit approximations of multi-dimensional partial differential equations. *SIAM Journal on Numerical Analysis*, 5:530–558, 1968.
11. D.C. Wilcox. *Turbulence Modeling for CFD.* DCW Industries, Inc., La Canada, 1993.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Cell-Vertex Multigrid Solvers in the PARAGRID framework

S. Sibilla[a] and M. Vitaletti[b]

[a]Aermacchi S.p.A, Air Vehicle Technology,
P.O. Box 101, 21040 Venegono Superiore, Italy

[b]IBM SEMEA S.p.A., ST&E - ECSEC,
P.le Giulio Pastore 6, 00144 Rome, Italy

FLO67P-2 is a parallel Euler solver based on the cell-vertex multigrid code FLO67 of A. Jameson. The solver is implemented as a module hosted within the PARAGRID framework, the latter providing multi-block management functions and automatic block-level parallelism on distributed systems. Some topics concerning a multi-block implementation of the multigrid algorithms are examined. The parallel efficiency of the whole code has been assessed on the IBM SP2 parallel system.

1. Introduction

Multiblock methods combine the simplicity and efficiency of CFD algorithms based on a single-block structured grid with the ability to handle geometrically complex regions. The latters are mapped by regular non overlapping blocks with arbitrary topological connections. Multiblock codes can run very efficiently in parallel mode by exploiting block-level parallelism. A significant progress in this area is achieved by implementing multiblock management functions — usually encapsulated in a specific application — within a separate, general purpose, framework.

1.1. PARAGRID

PARAGRID [1] [2] is a parallel multiblock framework developed at IBM ECSEC. A program solving a set of discretized field equations on a single-block structured grid can be integrated as a subdomain solver, so that multiple copies of the same code — each one computing a different grid block — can work concurrently on different nodes of a parallel system. An arbitrary number of field variables can be associated with the *grid sites* which are defined for each elementary grid *cell*, namely the cell center, the cell vertices and the cell faces. This generality is needed to support different discretization methods while *coarse* grid levels can be explicitely defined to ease the implementation of multi-grid and multi-level algorithms. The basic concept of PARAGRID is that of a *halo region* surrounding the *core region* of each subdomain grid:

- The evolution of the field is computed through a sequence of *update* steps performed by the hosted application module — in parallel — on all blocks.

- Updates include internal boundary nodes (cell-vertices). One difficulty comes from the existence of as many *replicas* of a boundary node as there are blocks sharing that particular site. Also, the grid around a boundary *edge* or *corner* is generally **unstructured**. Computing internal boundaries by *one-sided* algorithms generally causes the assignment of non identical values to the replicas of the same physical node as computed on different, contiguous blocks.

- Halo data automatically reflect the new status after a field update. Consistency of field data over all replicas of a boundary node is enforced by averaging.

This paper compares two different implementations of FLO67 within PARAGRID. FLO67 is a CFD Euler solver based on the Jameson [3] cell-vertex multigrid method where relaxation is achieved through an explicit 5-stage Runge-Kutta time-stepping algorithm with *implicit* residual smoothing.

1.2. FLO67P

The first attempt to integrate the FLO67 code within PARAGRID has been described in a previous work [2]. FLO67P — the modified code — achieved a maximum level of localization, the whole multigrid cycle being performed independently within each block with *frozen* halo data on all grid levels. FLO67P has minimum memory requirements and maximum parallel efficiency. However, the *one-sided* evaluation of interfaces leads to inconsistencies in the computation of the flow field on internal boundary nodes. For large time-steps, the automatic averaging performed *a posteriori* by PARAGRID is not sufficient to ensure convergence of field values for all replicas of an interface node.

2. FLO67P-2

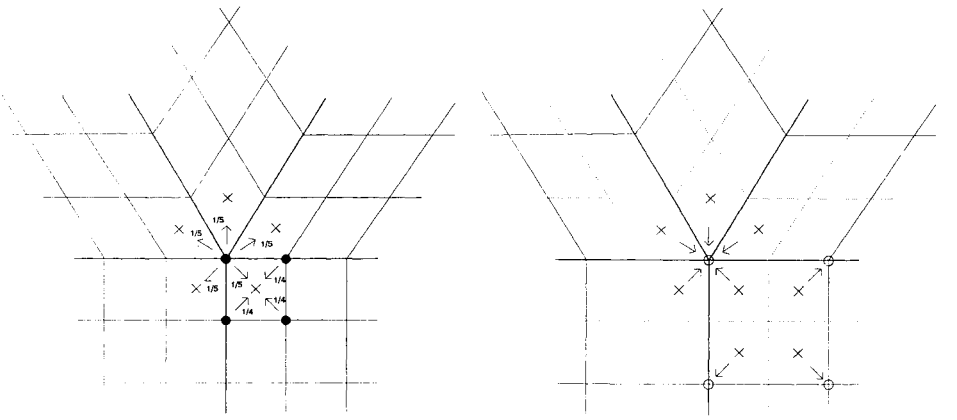
FLO67P-2 is a new version of the code in which departures from the serial algorithm are kept to a minimum, as they mainly originate from the adaptation of FLO67 to multi-block grids.

The control volume of a *cell-vertex* lying on a block *boundary edge* or *corner* generally consists of an *unstructured* collection of cells, with solid boundaries possibly inserted between cell faces. One of the goals of FLO67P-2 was to provide an accurate computation of the *convective fluxes* for control volumes of such general types.

A whole Runge-Kutta sweep of the Jameson algorithm is implemented, on a given grid level, as a sequence of 5 parallel update steps, one for each stage. The halo region of each block is updated with the most recent field values before starting the computation of each new stage, thus trying to ensure that identical field values be assigned to all replicas of a boundary vertex at the end of the whole sweep. Achieving this goal would actually require a *global* serial implementation of the *implicit residual smoothing* which is hard to generalize to arbitrary multi-block grids. This option would also adversely impact the parallel efficiency of the whole code. A similar difficulty was found with the computation of diffusive fluxes for the adaptive dissipation scheme. A full generalization to general multi-block grids requires to adopt an unstructured version of the scheme, which is still under investigation.

Presently, implicit residual smoothing is applied locally within each block. Also, *artificial dissipation* terms may be 'biased' by the computational stencil adopted in a particular

Figure 1. Collection of residuals to a coarser mesh.



block for grid nodes belonging to 'unstructured' edges or corners. Therefore, inconsistencies can still arise at internal boundaries when dealing with topologically complex grids and/or large time-step sizes. The problem is much less critical in FLO67P-2 than in the previous code, because the contribution of convective fluxes is fully consistent across block boundaries. Discrepancies among the replicas of a boundary node which are due to the 'block-biased' terms, are averaged out — on each grid level — after each stage of the RK sweep.

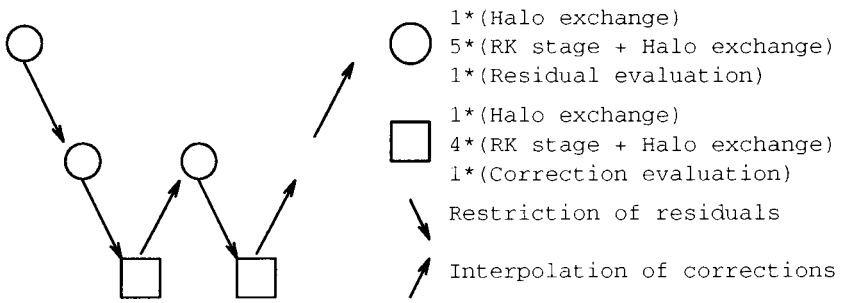
2.1. Multi-block multi-grid

Extending the multi-grid collection and interpolation procedures to a general multi-block topology is relatively easy within the new PARAGRID. In FLO67 total residuals are accumulated on cell vertices, and *residual collection* on a coarser mesh is accomplished in two steps, as shown in figure 1. On the fine mesh, the accumulated nodal values of the residuals are distributed to the neighboring cells. A node surrounded by N cells will donate $1/N$ of its residuals to any such cells, their number being readily available from PARAGRID. Cell residuals are then accumulated on the vertices of the coarser mesh.
No special provisions are necessary for the interpolation of coarse grid corrections. Nodal corrections are simply transferred to the finer meshes, where interpolation is then performed within each block.

2.2. Communication requirements

In order to quantify the impact of the new design on the network communication *latency* and *bandwidth* let take the case of a problem run with W *multi-grid cycles* over 3 grid levels, as illustrated in figure 2. A whole W-cycle required only 2 halo-updates in FLO67P (both on the finest grid level) while it takes 28 halo-updates in FLO67P-2. The latency overheads, in this example, are increased by a factor of 14, while the total amount of

Figure 2. W-cycle in FLO67P-2 (3 levels).



transferred data increased by a factor of about 10 (only 6 among the 28 halo updates run on the finest grid).

3. Numerical Experiments

The new parallel code went through a number of validation tests performed by Aermacchi [4]. The first test case involved the supersonic flow around a blunt body (cylinder closed by a sphere). Different **multiblock** splittings of a **single** block grid were used and results compared with those obtained with the serial code on the unsplitted grid. In figure 3 the flow field solution for a free-stream Mach number of 2 was obtained by the serial FLO67 code on the unsplitted grid and by FLO67P-2 on a 4-block splitting of the same grid. In both cases, the computation was run at a local Courant number of 6 with W multigrid cycles on 3 grid levels. The two solutions are identical. The solution obtained with FLO67P on the 4-block grid is also in good agreement with the single-block solution, but the residuals only decrease by 2 orders of magnitude (left of figure 4). In the figure, the plot grid spacing horizontally measures 25 time-steps, while it corresponds to a factor of 10 along the vertical, logarithmic scale. The convergence history includes 50 steps done on a coarse grid and 50 steps on a medium grid (multi-level initialization) before starting the finest grid calculations. The stopping of convergence in FLO67P was clearly caused by the loose coupling of the solutions at block interfaces. By contrast, very similar convergence histories were obtained by the serial code and FLO67P-2 (right).

A second test case is the transonic vortical flow around a wing-body-canard configuration. In this case, the grid is composed of 32-blocks (half configuration). The solution for a free-stream Mach number of 0.85 and an angle of attack of 10 degrees is illustrated in figure 5. The CP isolines on the upper surface of the body are illustrated on the left. Isolines of the total pressure losses are shown on the right — on a cross-flow section — to evidentiate the two strong vortices which span without distortion across inter-block boundaries.

Figure 3. CP iso-lines with FLO67 (left) and FLO67P-2(right)

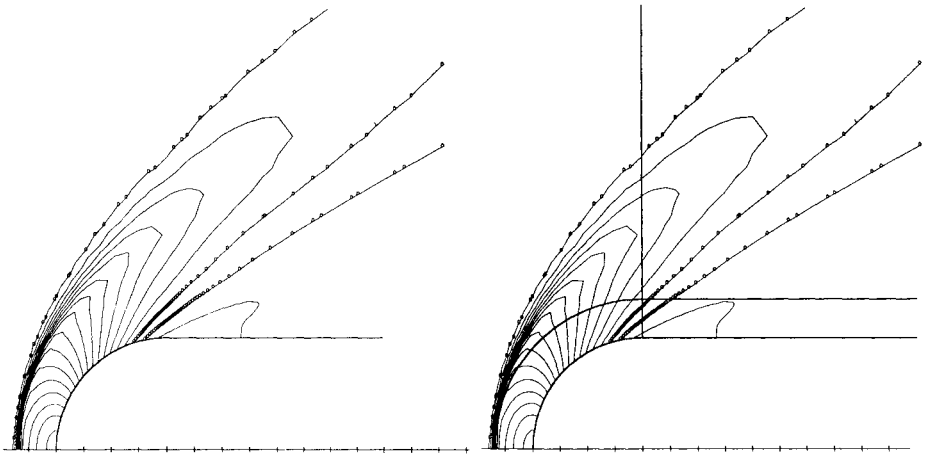


Figure 4. Max. density residual history (single precision). Left: FLO67P on a single-block grid. Right: FLO67P-2 on a 4-block splitting of the same grid.

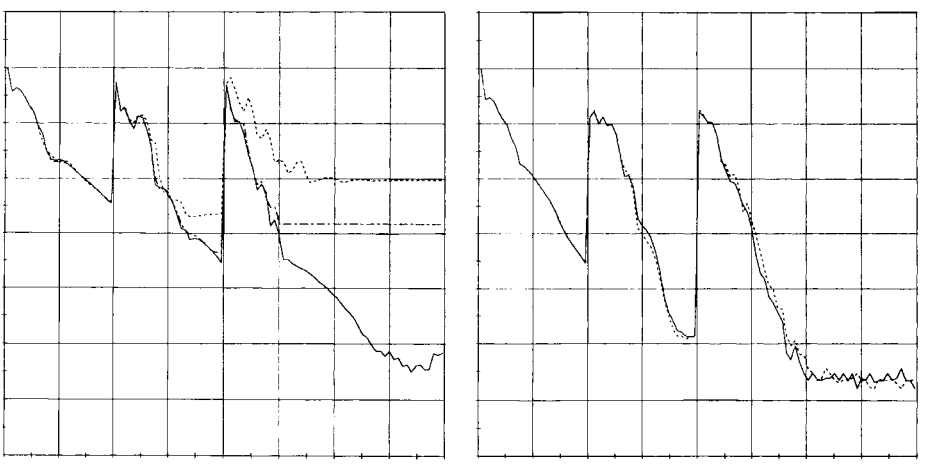
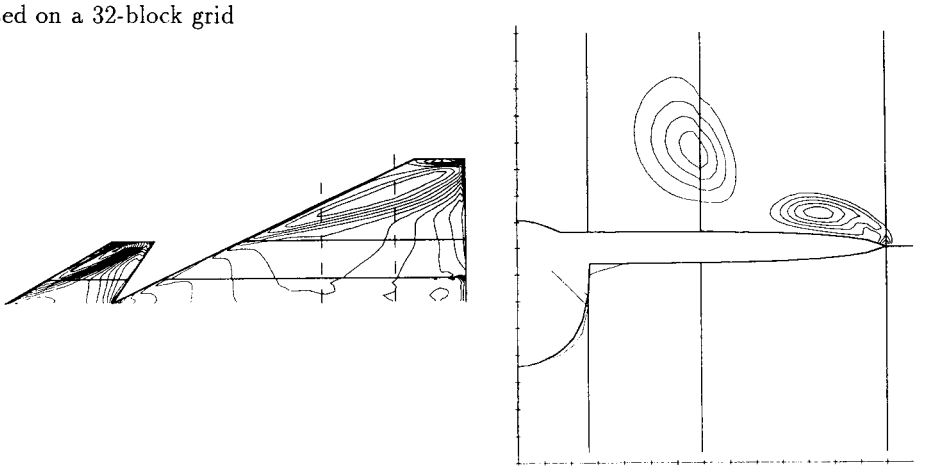


Figure 5. CP isolines (left) and pressure losses (right) for a transonic vortical flow computed on a 32-block grid

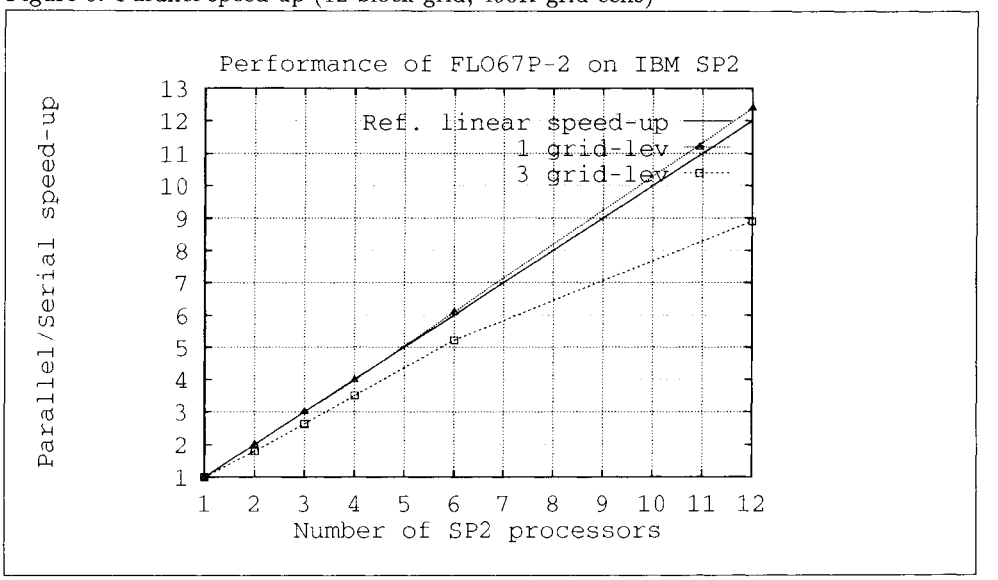


3.1. Parallel Efficiency on IBM SP2

The parallel efficiency and scalability of FLO67P-2 was measured on IBM SP2 parallel systems. A well-balanced computational load is the first requirement for a high parallel speed-up. PARAGRID features both automatic and user-specified load-balancing [1]. In the first case it is assumed that the computational cost is directly proportional to the number of grid cells in a block, a condition which is approximately satisfied by FLO67P-2. A direct specification can be tried by the user when blocks are of equal size and their number is a multiple of the available processors. The automatic algorithm works well when the number of blocks largely exceeds the number of available processors. A preliminary, automatic splitting of the largest grid blocks can be performed in a pre-processing stage if the previous condition is not satisfied. An automatic re-combination of results at the end of the computation can hide the complexity of the whole process to the end-user. A perfectly balanced test-case — a 12-block grid around a sphere — was used to ease the interpretation of efficiency measurements. The grid has a total number of 490 thousand cells. Tests were run on 1, 2, 3, 4, 6 and 12 processors, with and without multi-grid. Multi-grid runs used W cycles on 3 grid levels. The parallel speed-up is plotted in figure 6 for both series of tests. It is observed that runs without multi-grid scale well up to 12 processors, with a moderate super-linear speed-up, reasonably due to a better use of the cached memory system achieved through a greater localization of program data. The efficiency of multi-grid runs drops from 87% to 75% when doubling the number of processors from 6 to 12.

The PARAGRID communications were implemented on top of the IBM MPL message passing library, using the bi-directional (send/receive) primitive. This ensures an optimal use of the IBM SP2 High-Performance Switch. The point-to-point bandwidth, measured at the application level, ranges from 20 to 40 megabytes/sec, depending on the message length.

Figure 6. Parallel speed-up (12-block grid, 490K grid cells)



4. Conclusions

An accurate implementation of explicit multi-stage, multi-grid CFD algorithms poses strong requirements on the communication latency and bandwidth of a distributed memory system which cannot be met on ordinary clusters. The good performance results obtained with FLO67P-2 on the IBM SP2, a distributed parallel system featuring a high-performance switch, suggest that scalability to tens of processors can be achieved, on this type of codes, for medium and large problems. Based on the present results, each processor should be assigned no less than 40 thousands grid cells.

REFERENCES

1. F. Dellagiacoma, S. Paoletti, F. Poggi, and M. Vitaletti. Multidomain computations of compressible flows in a parallel scheduling environment. In R.B. Pelz, A. Ecer, and J. Hauser, editors, *Parallel Computational Fluid Dynamics '92*, pages 111–122, New Brunswick,NJ,USA, May 1992. North-Holland.
2. F. Dellagiacoma, M. Vitaletti, A. Jameson, L. Martinelli, S. Sibilla, and L. Visintini. Flo67p: a multi-block version of flo67 running within paragrid. In *Parallel Computational Fluid Dynamics '93*, Paris,France, May 1993. North-Holland.
3. A. Jameson. A vertex based multigrid algorithm for three dimensional compressible flow calculations. *ASME Symposium on Numerical Methods*, 1986.
4. S. Sibilla. Flo67p-2 euler solver for multi-block structured grids of general topology. Technical report, 1994. Aermacchi Report 567-ANG-378.

This Page Intentionally Left Blank

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

FLUID FLOW IN AN AXISYMMETRIC, SUDDEN-EXPANSION GEOMETRY

C.F. Bender, P.F. Buerger, M.L. Mittal, and T.J. Rozmajzl
Program for Computational Reactive Mechanics
Ohio Supercomputer Center
1224 Kinnear Road, Columbus, OH 43212, U.S.A.

This paper deals with fluid flow in a sudden expansion geometry for moderately high Reynolds number ≈ 2500. The full time dependent Navier-Stokes equations are solved in this geometry. To resolve all the prevailing excited scales of the fluid flow, a fine staggered grid is used. The fine mesh and small time steps necessitate computation on scalable parallel machines. The performance of parallel implementation and computations is discussed.

1. Introduction

Fluid flow in sudden expansion geometries, with or without solid particles, is an important technological process for many industrial and scientific applications, especially for combustion processes. For such geometries, even with moderately high values of the Reynolds number, the separated boundary-layer becomes unstable, and the flow exhibits repeated vortex formation and shedding at the corner point near the sudden expansion. The formation and persistence of large scale coherent vortex structures within turbulent wake-like regions give rise to turbulent behavior.

The numerical simulation of such a turbulent fluid flow that includes details such as separation phenomena and reverse flows precludes the use of the Reynolds averaged Navier-Stokes equations or modeling approaches such as mixing lengths and $\kappa - \epsilon$. Subgrid-scale modeling continues to be a weak part of the large eddy simulation (LES) technique. Spectral methods can deal with only simple square or cube geometries. One alternative is the Direct Numerical Simulation (DNS), in which the full Navier-Stokes equations are solved numerically. The DNS technique is relatively new and has been developed in the last two decades [1,2]. DNS is amenable for parallel computing. The challenge is to devise computational techniques that can exploit the latest advances in computer system software and hardware.

A full time-dependent, Navier-Stokes analysis is used to simulate fluid flow and particle motion in an axisymmetric sudden expansion geometry for Reynolds number (Re) = 2500. The formulation, based on global conservation of mass and momentum, is given by Osswald et al. [3]. A vorticity stream function formulation, instead of one based on the primitive variables, is used for this flow simulation. Even with transformation to generalized orthogonal curvilinear coordinates, the strong-conservation form is obtained for both the vorticity-transport equation and the elliptic stream function equation.

A clustered conformal coordinate procedure establishes a surface-oriented orthogonal coordinate system with grid clustering/stretching characteristics and a novel grid-point placement in the physical flow domain. More technical details of this formulation are in [3] and [4]. The governing equation for vorticity is solved by an alternating-direction fully implicit (ADI) method in time, using a forward time marching scheme. The corresponding stream function distribution is obtained by a fully implicit solution of the elliptic stream function equation.

The flow boundary conditions are derived from symmetry across the centerline and zero slip at the walls. The streamwise asymptotic forms of the governing equations are solved to provide the inflow/outflow boundary conditions; these conditions maintain consistency between the boundary values and the interior solution. The time evolution of the flow field is pursued long enough to achieve the quasi-steady state.

These simulations need a large number of grid points in space to resolve all the prevailing excited scales of the fluid flow and a proper time step to ensure flow numerical diffusion. A grid of 512 points in the axial direction and 128 points in the radial direction is used with time step $\Delta t = 4.0 \times 10^{-6}$ in the present simulation. The resulting system of equations is solved by the *GMRES* iterative method to minimize the effects of roundoff errors.

The trajectories of the particles of different sizes, injected at different radial locations in such a quasi-steady state flow, can be computed using Lagrangian equations. The particle motion occurs due to the drag force from the local fluid velocity.

The separation point, the recirculating flow, and the large-scale vortex structure for the fluid flow are shown and discussed. We will also discuss the roundoff errors introduced by the large number of mesh points and the need to maintain numerical accuracy.

2. Governing equations

The unsteady, incompressible Navier-Stokes equations, in terms of the vorticity vector $\vec{\omega}$ and the velocity vector $\mathbf{V}$, consist of a temporally parabolic vorticity transport equation

$$\frac{\partial \vec{\omega}}{\partial t} + (\mathbf{V} \cdot \nabla)\vec{\omega} = (\vec{\omega} \cdot \nabla)\mathbf{V} - \frac{1}{Re}(\nabla \times \nabla \times \vec{\omega}) \tag{1}$$

together with the kinematic definition for vorticity

$$\nabla \times \mathbf{V} = \vec{\omega} \tag{2}$$

For the present axisymmetric flow, involving only two spatial coordinates, a stream function ψ is defined as

$$\mathbf{V} = \nabla\psi \times \vec{e^3}, \tag{3}$$

which also satisfies the continuity equation

$$\nabla \cdot \mathbf{V} = 0 \tag{4}$$

The vorticity vector $\vec{\omega}$ and ψ are related as

$$\vec{\omega} = \nabla \times \mathbf{V} = \nabla \times \nabla\psi \times \vec{e^3} \tag{5}$$

where $\vec{e^3}$ is the local contravariant base vector normal to the (ξ^1, ξ^2) coordinate surface as shown in figure 1. Here, ξ^1 and ξ^2 represent generalized orthogonal curvilinear coordinates for an axisymmetric geometry.

2.1. Coordinate transformation and the grid

A generalized curvilinear coordinate transformation **T** is defined to transform the flow configuration from the physical cartesian coordinates (x^1, x^2, x^3) to the computational coordinates (ξ^1, ξ^2). Figure 1 shows the coordinate transformation **T**.

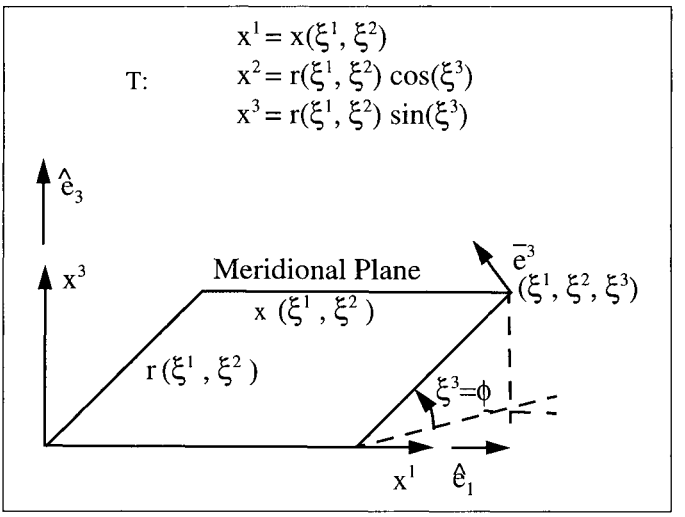


Figure 1. Generalized axisymmetric coordinate transformation T

An analytically defined *clustered conformal* coordinate procedure is used to establish the surface-oriented orthogonal coordinates with desirable grid clustering characteristics throughout the physical flow domain. A boundary-aligned conformation coordinate transformation T_H defines the boundary. A grid clustering transformation T_C is then superimposed over T_H for proper flow resolution. The *clustered conformal* coordinate transformation T=T_H.T_C maps the unit square $[\xi^1, \xi^2]$ in the computational plane onto a doubly-infinite step configuration in the physical plane. A uniformly spaced computational mesh is transformed into the clustered cordinate grids. The details of the grid generation and coordinate transformation are given in [3].

3. Numerical solutions

For generalized axisymmetric flow,

$$\vec{\omega} = \omega^3 \vec{e_3}, \tag{6}$$

where ω^3 is the contravariant component of the vorticity, and $\vec{e_3}$ is the local covariant base vector normal to the (ξ^1, ξ^2) coordinate surface. Using equations 6 and 3 in equation

1, we write the vorticity transport equation as

$$\sqrt{g}\frac{\partial\omega^3}{\partial t}+\frac{\partial}{\partial\xi^1}\left(\omega^3\frac{\partial\psi}{\partial\xi^2}\right)-\frac{\partial}{\partial\xi^2}\left(\omega^3\frac{\partial\psi}{\partial\xi^1}\right)$$
$$=\frac{1}{Re}\left[\frac{\partial}{\partial\xi^1}\left(\frac{g_{22}}{\sqrt{g}}\frac{\partial}{\partial\xi^1}(g_{33}\omega^3)\right)+\frac{\partial}{\partial\xi^2}\left(\frac{g_{11}}{\sqrt{g}}\frac{\partial}{\partial\xi^2}(g_{33}\omega^3)\right)\right] \tag{7}$$

The equation for ψ is obtained from equations 3 and 5.

$$\frac{\partial}{\partial\xi^1}\left(\frac{g_{22}}{\sqrt{g}}\frac{\partial\psi}{\partial\xi^1}\right)+\frac{\partial}{\partial\xi^2}\left(\frac{g_{11}}{\sqrt{g}}\frac{\partial\psi}{\partial\xi^2}\right)=-\sqrt{g}\omega^3. \tag{8}$$

Here, g represents the determinant of the covariant metric tensor g_{ij}, where

$$g_{ij}=\sum_{k=1}^{3}\left(\frac{\partial x^k}{\partial\xi^i}\right)\left(\frac{\partial x^k}{\partial\xi^j}\right) \tag{9}$$

3.1. Boundary and initial conditions

The centerline symmetry boundary conditions are used for ψ and ω. Thus

$$\psi_c=0 \tag{10}$$

$$\omega_c^3=\omega_p^3/\sqrt{g_{33}}=\text{finite} \tag{11}$$

as ω_p^3=0 and $\sqrt{g_{33}}=0$ at the centerline.

The No-slip boundary condition on the solid wall gives:

$$\psi_w=0.5 \tag{12}$$

and

$$\frac{\partial}{\partial\xi^2}\left(\frac{g_{11}}{\sqrt{g}}\frac{\partial\psi}{\partial\xi^2}\right)\Big|_{wall}=-\sqrt{g}\omega^3\Big|_{wall} \tag{13}$$

subject to the constraint

$$\frac{1}{\sqrt{g}}\frac{\partial\psi}{\partial\xi^2}\Big|_{wall}=0 \tag{14}$$

Along the inflow and outflow boundaries at upstream and downstream infinity, the flow becomes diffusion dominated in the normal direction. Hence, the asymptotic forms of equations 5 and 8 are used as the inflow and outflow conditions. This procedure maintains consistency between the boundary values and the interior solution. This gives

$$\frac{\partial\omega^3}{\partial t}=\frac{1}{Re\sqrt{g_{22}g_{33}}}\frac{\partial}{\partial\xi^2}\left(\frac{1}{\sqrt{g_{22}g_{33}}}\frac{\partial}{\partial\xi^2}(g_{33}\omega^3)\right) \tag{15}$$

$$\frac{1}{\sqrt{g_{22}g_{33}}}\frac{\partial}{\partial\xi^2}\left(\frac{1}{\sqrt{g_{22}g_{33}}}\frac{\partial\psi}{\partial\xi^2}\right)=-\omega^3 \tag{16}$$

Initial conditions are obtained from the steady-state solution of equation 16. The initial vorticity distribution is obtained as a solution of equation 8 with appropriate boundary conditions. This implies a flow starting impulsively from rest. Details of this procedure are provided in [3].

3.2. Numerical methods

A second-order accurate central finite difference scheme is used for spatial discretization. Starting from the initial state, the vorticity field is calculated with a forward time marching scheme. The computational problem is divided into a number of small parallel jobs by using an alternating direction implicit (ADI) method. Using ADI, we get a tridiagonal system of matrices for each direction that can be solved in parallel. For the present problem, there are 128 tridiagonal matrices of size (512×512) for the axial direction and 512 matrices of size (128×128) for the normal direction. The work load for each of these matrices is also approximately the same, which makes it easy to balance the load on the machine. These tridiagonal matrices are solved in a sequential manner, one each on a single processor.

Because of the large matrix size and the need to solve the equations repeatedly for a sufficiently long time to achieve the quasi-steady state of the turbulent flow, it is important to control the roundoff errors. Direct solution methods could be more efficient computationally, but with these methods, the residual error can also be as high as 10^{-1}. The roundoff error is controlled by using the *GMRES* [5] with the criterion that the root mean square of the residual error is always less than 10^{-4}. To accelerate the convergence, the diagonal matrix is used as the preconditioner. The elliptic stream function equation results in a block tridiagonal system after discretization. A parallel implementation of the direct fully implicit solution of the block tridiagonal matrix is used to solve the stream function equation.

We intend to use the ADI method [6,7] for the solution of the Poisson equation also to improve the overall performance. Alternatively, we may use the algorithm of Hajj and Skelboe [8] to parallelize the Block Tridiogonal matrix system.

4. Physical results

Figures 3 and 4 give the vorticity contours of the flow at the nondimensional time $t = 3.7$. The flow is highly turbulent in the upstream region whereas in the downstream region the large scale structures can be seen. As these large eddies interact, the flow becomes turbulent. The large scale structures are shown in figure 4 at the nondimensional axial length between 5.0 and 6.0, and figure 3 shows the vortices and turbulent character of the flow near the sudden expansion region. The streamline contours of the flow, the separation points, and the recirculation zones in the left corner of the geometry are shown in figures 5, 6, 7, and 8. The recirculation zone, as well as the separation length increase with time until a quasi-steady state can be reached. At nondimensional times $t = 0.04, 0.135, 0.19,$ and 0.2,the lengths are ≈ 0.65, 0.9, 1.12, and 1.15, respectively, as seen in the streamline contour figures. At time $t = 3.35$, it is ≈ 5.5, and at $t = 3.75$ it is 6.25. For turbulent quasi-steady state flows, the separation length will oscillate. As a result of the separation length increasing, the recirculation zone also increases, as can be seen in figures 5, 6, and 7.

5. Parallel implementation and performance

To measure the performance of these computations, the *speedup* $S_p(np)$ and the *efficiency* $E_p(np)$ are defined as:

$$S_p(np) = \frac{T(1)}{T(np)} \; ; \; E_p(np) = \frac{S_p(np)}{np} = \frac{T(1)}{np \times T(np)},$$

where np denotes the number of processors and T(np) is the computational time for np processors.

These algorithms were first implemented on a Cray Y-MP, a multiprocessor, shared-memory vector supercomputer. Using seven processors of the Y-MP, we achieved a speedup of 6.55 for an efficiency of 94%.

These algorithms are now implemented on a Cray T3D, a scalable parallel supercomputer, using CRAFT compiler directives. Single processor element (PE) performance, especially making effective use of the data and instruction caches, is crucial for the overall performance of the T3D. Using several optimization techniques such as private arrays with dimensions not multiples of 1024, loop unrolling, padding, etc. we achieve a speedup of 17.9 for an efficiency of 56% on 32 PEs. Table 1 and figure 2 show the overall performance of the code on the T3D for one through 32 PEs. Memory limitations prevented our running the code on a single PE. The timings for a single PE shown in Table I and figure 2 are extrapolated from the timings for two PEs.

With CRAFT operations we achieve approximately 2.5 - 3.0 Mflops per PE. We hope to improve the single PE performance and achieve 10 Mflops per PE through the use of routines from the shared-memory or MPI libraries.

Table 1
Performance on CRAY T3D with CRAFT

np	Grid points/PE	Time(minutes)	Speedup	Efficiency
1	65536	260.3	1	100%
2	32768	130.2	2	100%
4	16384	68.5	3.8	95.0%
8	8192	37.6	7.0	87.5%
16	4096	22.3	11.7	73.0%
32	2048	14.6	17.9	56.0%

Single PE performance is based on 2 PEs performance.

6. Conclusions

The performance of the parallel computations of the fluid flow for the present application is very encouraging as seen from the speedup curve, although Mflops and per processor performance is well below the optimal. With shared memory operations, we hope this performance can be improved and Gigaflops performance can be achieved. With this fluid flow, we can now calculate the trajectories of solid particles.

REFERENCES

1. R.S. Rogallo and P. Moin, Ann. Rev. Fluid Mech. 16 (1984) 99
2. W.C. Reynolds, *Whither Turbulence? Turbulence at the Crossroads* J.L. Lumley (ed), Springer-Verlag, New York, 1990
3. G.A. Osswald, K. N. Ghia, and U. Ghia, Unsteady Navier-Stokes Simulation of Internal Separated Flows Over Plane and Axisymmetric Sudden Expansions. *AIAA Paper 84-1584*
4. M.L. Mittal and U. Ghia, *Proceedings of Combustion Fundamentals and Applications, 1991 Spring Technical Meeting, Central State Section, The Combustion Institute* (Nashville, Tennessee, April 22-24), 65-71
5. Z. Bai, D. Hu, and L. Reichel, IMA Journal of Numerical Analysis 14 (1994) 563
6. D.W. Peaceman and H.H. Rachford Jr., J. Soc. Indust. Appl. Math. 3 (1955) 28
7. A. Hadjidimos, Numer. Math. 13 (1969) 396
8. I.N. Hajj and S. Skelboe, Parallel Computing 15 (1990) 21

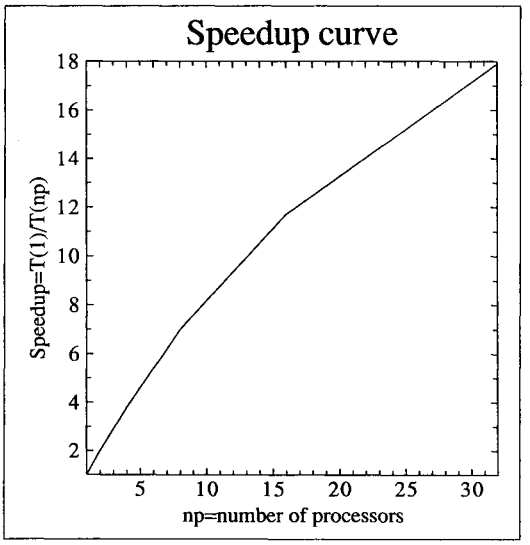


Figure 2. Speedup curve for 32 processors

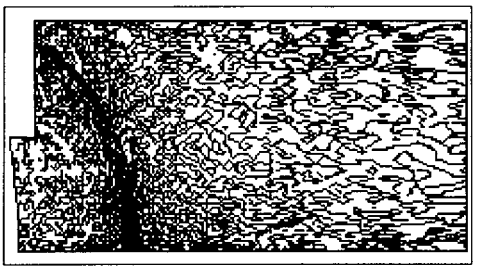


Figure 3. Vorticity contours upstream

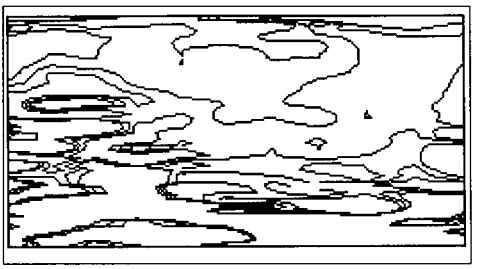


Figure 4. Vorticity contours downstream

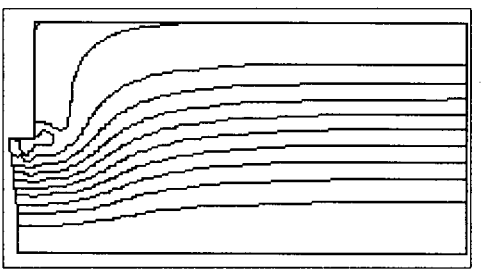


Figure 5. Streamline contours at nondimensional time t = 0.04

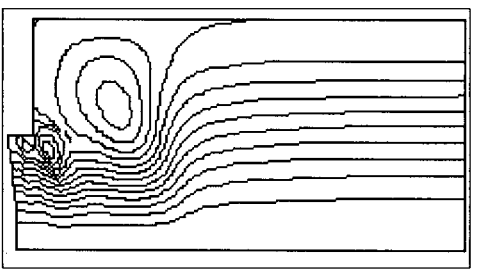


Figure 6. Streamline contours at nondimensional time t = 0.135

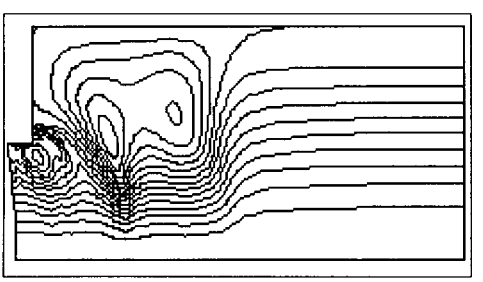


Figure 7. Streamline contours at nondimensional time t = 0.19

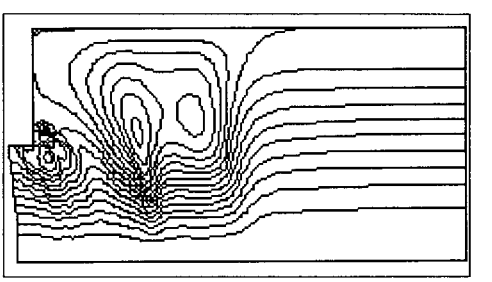


Figure 8. Streamline contours at nondimensional time t = 0.20

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Implicit Navier-Stokes Codes in Parallel for Aerospace Applications

K.J.Badcock and B.E.Richards[a*]

[a]Department of Aerospace Engineering, University of Glasgow, Glasgow, G12 8QQ, U.K.

The parallel implementation of implicit codes to simulate two and three dimensional compressible and turbulent fluid flows is discussed. The implicit methods are based on a preconditioned conjugate gradient type solution of the large and sparse linear system at each time step. The preconditioner is based on the alternating direction implicit factorisation of the linear system. An efficient parallel implementation is achieved by using data transposition to locate the data relating to complete mesh lines in each coordinate direction onto a single processor at the appropriate time in the calculation. Results are given to illustrate the performance achieved on the Cray T3D and a workstation cluster for the transonic and turbulent flow over an aerofoil section and a wing.

1. Introduction

It is becoming possible to accurately simulate complex three-dimensional flows due to advancements in both computer and algorithm power. Parallel computing can provide supercomputing at a modest cost through workstation clusters or a potential performance approaching gigaflops through machines such as the Cray T3D. To use these machines effectively careful attention must be paid to how the algorithm can best exploit the distributed processing.

Implicit methods are currently preferable for the simulation of some types of fluid flow due to better stability properties when compared with explicit methods. Unsteady viscous flows are one example. The solution of the large sparse linear system at each implicit time step presents a particular problem for the parallel implementation of implicit methods. Direct methods are not well suited to parallel implementation because Gaussian elimination and back substitution are essentially sequential in nature. Iterative methods such as conjugate gradient type algorithms are more promising since their main computational work is based on a matrix-vector product which can be carried out efficiently in parallel. However, effective preconditioning is required for iterative methods and this is usually based on a direct solution of a simplified system, eg incomplete LU decomposition or alternating direction implicit (ADI) factorisation.

The use of the ADI factorisation as a preconditioner for a conjugate gradient squared (CGS) solution of the linear system at each implicit step of a solution of the Navier-Stokes equations is described in [1]. A novel two factor method for three-dimensional flows is

*The workstation cluster used in this study was purchased as part of a project funded by JISC under the New Technologies Initiative. Time on the Edinburgh Parallel Computing Centre Cray T3D was obtained as part of the Computation of Complex Aerodynamic Flows consortium.

discussed in [2]. Care must be taken in the parallel implementation of these methods due to the sequential nature of the ADI preconditioning used. However, the preconditioning decouples between lines in the different coordinate directions during the calculation and data transposition can be used to shuffle data around so that each processor can complete the calculation on a mesh line independently of other processors. In this way communication is concentrated in several intensive bursts, latency is eliminated and high parallel efficiencies are obtained.

The details of this approach are discussed in this paper. In the following section the numerical methods are briefly described. Then the parallel implementation of the methods for aerofoil flows and wing flows is discussed. Code timings are given for several machines and issues such as the effectiveness of the parallel implementation are considered.

2. The Numerical Methods

2.1. Aerofoil Flows

The AFCGS method for steady and unsteady aerofoil flows has been presented in [1] and [3] respectively and the reader is referred therein for full details. The motivation behind the approach is to use conjugate gradient methods to remove factorisation error effects from implicit methods. The Alternating Direction Implicit (ADI) approximate factorisation, which is frequently used to provide a solution to the large linear system which arises at each implicit time step, is used as a preconditioner for the conjugate gradient solution of this linear system. A brief description of the method is now given.

The thin-layer Reynolds' Averaged Navier-Stokes equations in generalised curvilinear co-ordinates (ξ, η) with η normal to the surface can be denoted in non-dimensional conservative form by

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\partial \mathbf{f}}{\partial \xi} + \frac{\partial \mathbf{g}}{\partial \eta} = \frac{\partial \mathbf{s}}{\partial \eta} \tag{1}$$

where $\mathbf{w}$ denotes the vector of conserved variables, $\mathbf{f}$ the convective streamwise flux, $\mathbf{g}$ the convective normal flux and $\mathbf{s}$ the normal viscous flux.

One implicit step can be written as

$$(I + \Delta t \frac{\partial \mathbf{R}_\xi}{\partial \mathbf{w}}^n + \Delta t \frac{\partial \mathbf{R}_\eta}{\partial \mathbf{w}}^n)\Delta \mathbf{w}^n = -\Delta t(\mathbf{R}_\xi^n + \mathbf{R}_\eta^n) \tag{2}$$

where $\mathbf{R}_\xi$ and $\mathbf{R}_\eta$ are terms arising from the spatial discretisation in the ξ and η directions respectively and

$$\frac{\partial \mathbf{f}}{\partial \xi} \approx \mathbf{R}_\xi \qquad \frac{\partial (\mathbf{g} - \mathbf{s})}{\partial \eta} \approx \mathbf{R}_\eta.$$

$$\Delta \mathbf{w} = \mathbf{w}^{n+1} - \mathbf{w}^n$$

In the present work the spatial terms are discretised using Osher's flux approximation with MUSCL interpolation and the Von Albada limiter for the convective terms and central differencing for the viscous fluxes. The Baldwin-Lomax turbulence model is employed to provide a turbulent contribution to the viscosity.

The ADI factorisation of equation (2) is

$$(I + \Delta t \frac{\partial \mathbf{R}_\xi}{\partial \mathbf{w}}^n)(I + \Delta t \frac{\partial \mathbf{R}_\eta}{\partial \mathbf{w}}^n)\Delta \mathbf{w}^n = -\Delta t(\mathbf{R}_\xi^n + \mathbf{R}_\eta^n). \tag{3}$$

Denoting the linear system to be solved at each time step by

$$A\mathbf{x} = \mathbf{b} \tag{4}$$

we seek an approximation to $A^{-1} \approx C^{-1}$ which yields the system $C^{-1}A\mathbf{x} = C^{-1}\mathbf{b}$ more amenable to conjugate gradient methods. The ADI method provides a fast way of calculating an approximate solution to equation (4) or, restating this, of forming the matrix vector product $\mathbf{x} = C^{-1}\mathbf{b}$. Hence, if we use the inverse of the ADI factorisation as the preconditioner then multiplying a vector by the preconditioner can be achieved simply by solving a linear system with the right-hand side given by the multiplicand and the left hand side matrix given the approximate factorisation. The factors in C can be put in triangular form once at each time step with the row operations being stored for use at each multiplication by the preconditioner.

2.2. Wing Flows

The extension of the method to three-dimensions is complicated by two considerations. First, computer storage becomes a limiting factor due to the need to store large Jacobian matrices. Secondly, the three factor ADI factorisation in three-dimensions is significantly worse than in two-dimensions, making its use as a preconditioner less favourable. This fact however means that there are increased gains to be made in three dimensions by the use of an alternative to ADI.

One step of the method considered can be written as

$$(I + \Delta t \frac{\partial \mathbf{R}_x}{\partial \mathbf{w}} + \Delta t \frac{\partial \mathbf{R}_y}{\partial \mathbf{w}})(I + \Delta t \frac{\partial \mathbf{R}_z}{\partial \mathbf{w}})\delta \mathbf{w} = \mathbf{R}_{exp} \tag{5}$$

where $\mathbf{R}_{exp} = -\Delta t(\mathbf{R}_x + \mathbf{R}_y + \mathbf{R}_z)$. This two factor step can be loosely described as unfactored in each spanwise slice and approximately factored in the spanwise direction. A stability analysis [4] has shown that the method has similar stability properties to the two factor ADI method in two-dimensions. This represents a large improvement on the behaviour of the three factor method in three-dimensions. The linear system resulting from the first factor in equation 5 has a more complicated structure than the block pentadiagonal systems which are encountered for the three factor method. However, this sytem can be solved using a direct generalisation of the method described for two dimensions above i.e. we solve the system $C^{-1}A\mathbf{x} = C^{-1}\mathbf{b}$ by the CGS method where

$$A = (I + \Delta t \frac{\partial \mathbf{R}_x}{\partial \mathbf{w}} + \Delta t \frac{\partial \mathbf{R}_y}{\partial \mathbf{w}}), \tag{6}$$

$$C = (I + \Delta t \frac{\partial \mathbf{R}_x}{\partial \mathbf{w}})(I + \Delta t \frac{\partial \mathbf{R}_y}{\partial \mathbf{w}}) \tag{7}$$

and $\mathbf{b} = -\Delta t(\mathbf{R}_x + \mathbf{R}_y + \mathbf{R}_z)$. The two factor method has substantially reduced memory requirements compared with the fully unfactored method since only the matrix for one spanwise slice or one line in the spanwise direction need be stored at any one time.

A more detailed discussion of this method can be found in [2].

3. Parallel Implementation of Aerofoil Code

The major obstacle to an efficient parallel implementation of the AFCGS method is the inherently sequential nature of the ADI solution. This was overcome in [5] by using a transposition of the data to allow complete ADI sweeps to proceed independently on each processor. We use this approach here although extra communication is required for the present method because of the matrix-vector products required in the CGS algorithm.

The computational space is mapped onto the nodes by grouping complete mesh lines in both the ξ (streamwise) and the η (normal) directions onto a single node. Care has to be taken to make sure that η lines on either side of the wake cut are mapped to the same processor. The computation then falls into three phases. First, the matrix is generated and the factors are put in upper triangular form. The next phase is the multiplication of a vector by the matrix during the CGS solution and finally we have multiplication of a vector by the preconditioner which reduces to back substititution on the triangular factors of the ADI factorisation. For each phase data is held on a node for complete lines in one direction in the mesh and the entire computation relating to that direction is completed. The data is then communicated so that information for complete lines in the other direction is held on a single node and the computation for that direction proceeds.

For the calculation of the matrix, the solution from the previous time step is initially held on nodes by lines in the η direction. The fluxes and derivatives are calculated for each line and the contributions to the residual are formed. At this stage the factor

$$I + \Delta t \frac{\partial R_\eta}{\partial w} \tag{8}$$

is put in upper triangular form. The solution and the residual are then transposed so that each node holds information on complete lines in the ξ direction. No communication of the components of the matrix $\partial R_\eta / \partial w$ is made since these are stored and used on the nodes on which they are calculated. The calculation proceeds so that we have the left hand side of 2 stored in complete ξ and η lines on a node and the solution at time n and residual stored on complete ξ lines only.

Next, one step of ADI is performed to obtain a starting solution for CGS. This is achieved by performing ADI sweeps in the ξ direction first, then transposing the solution obtained and performing sweeps in the η direction to yield the ADI solution. When a multiplication by the preconditioner is required, this is performed in the same way.

The CGS solution proceeds with inner products and matrix multiplications being required. Contributions to an inner product are calculated locally on each node and are then either transmitted to a chosen node for summing or are summed using a global sum function. A matrix multiplication is achieved by finding the contribution from terms arising from the current storage direction direction (η), transposing the multiplicand and the partial product and then adding the terms arising from the multiplication in the other direction.

The parallel code was tested on the intel Hypercube at the SERC Daresbury Laboratory with the native message passing fortran routines being used. The test case involved the turbulent flow over a pitching aerofoil. The algorithm speeds using varying numbers of nodes on the intel Hypercube are shown in table 1 and indicate that the transpositions degrade the parallel efficiency. However, this is the case for most parallel preconditioned

Machine	CPU time	efficiency
4 nodes	1161	1.00
8 nodes	658	0.88
16 nodes	422	0.69

Table 1
Algorithm speeds in μsec/grid point/time step for an increasing number of nodes on the Hypercube. The parallel efficiency is based on the CPU time for 4 nodes since the problem is too large in terms of memory to fit onto one node.

conjugate gradient solvers and the efficiencies are good compared to those achieved eg using the incomplete ILU-conjugate gradient solver of [6]. It is anticipated that the efficiency will increase on finer meshes as the amount of computation relative to the communication increases on each node. In addition, the storage requirements of the algorithm have been divided amongst the processors. The breakdown of the speed up obtained by using 8 nodes of the Hypercube compared to a Sun Sparcstation 10 Model 30 for each main phase of the calculation is shown in table 2. The net speed up achieved by using the parallel machine is 3.2. The main computational work is involved with calculating the Jacobian and putting the ADI factors into triangular form. It is clear that the parallelisation is most effective for this phase of the calculation which takes up about eighty percent of the total work. The second most intensive part of the work is the CGS solution and this too is effectively parallelised even though the ADI preconditioning slightly degrades the efficiency. Finally, an ADI step is relatively inefficient due to the large communication time relative to the computational time involved.

The parallel code was also implemented on a cluster of Silicon Graphics Indy workstations at the University of Glasgow. The message passing was accomplished by using PVM version 3.3. The comparison of algorithm speeds (time in μsec/grid point/time step) on the Hypercube and on the workstation cluster is shown in table 3. It is clear that excellent performance on the workstation cluster is achieved compared with the Hypercube. This is due to the faster individual processors of the workstations. It should be noted that comparative speed-ups for the Hypercube and the cluster are not available since the code requires too much memory to run on one node of the Hypercube. However, the loss in potential performance due to the parallel processing does not seem to be any greater for the workstation cluster than on the Hypercube.

4. Parallel Implementation of the Wing Code

The three-dimensional algorithm has two distinct phases. First, there is the generation and solution of the large linear system in equation 6 arising from each spanwise slice of the mesh. Secondly, there is the solution of the banded linear systems arising from the second factor in the spanwise direction.

The first phase is split between processors in two ways. First, the spanwise sections are split into groups. Each group is then assigned to a set of processors with each spanwise

Part of Calculation	ratio of parallel:serial execution speed
total	3.2
Jacobian calculation	3.5
ADI step	1.5
CGS solution	3.1

Table 2
Comparison of speed-ups on 8 nodes of the Hypercube compared to a Sun Sparcstation 10 model 30 for various parts of the calculation.

Machine	algorithm speed	efficiency
SGI cluster 1 nodes	958	1.00
SGI cluster 6 nodes	230	0.69
SGI cluster 8 nodes	194	0.62
Hypercube 8 nodes	658	NA

Table 3
algorithm speeds in μsec/grid point/time step for various parallel machines. The parallel efficiency for the Hypercube is unavailable since the problem is too large in terms of memory to fit onto one node.

slice in the group being treated in a similar way to the two dimensional algorithm described above by those processors. The communication between the different groups of processors, each treating a different set of spanwise slices, is simply that which would be required by an explicit method so that the contributions to the residual (or the right-hand-side of the linear system) from the spanwise fluxes at the interfaces between the spanwise groupings can be evaluated. Since there is significantly less communication involved at this stage than is required to solve a spanwise slice in parallel, it is clear that the most efficient partition of the problem will arise when as large a number of spanwise groups as possible is used. For a fixed number of total processors this will reduce the number of processors which operate on a spanwise section.

The second phase of the calculation involves assigning complete spanwise lines in the mesh to single processors. Again, a transposition of the data is used so that the calculation involving a single line can proceed on a single processor without further communication. Once the updates are available a second transposition is used to restore storage by spanwise slices for the next time step.

The method has been implemented on the Hypercube under native message passing, on the Cray T3D at the Edinburgh Parallel Computing Centre under PVM v3.3 and on the workstation cluster described above under PVM v3.3. These codes were tested on a steady flow over an unswept wing with a NACA64A010 aerofoil section. The grid in this case contains 70000 points and is small for this type of application and hence the parallel

Machine	algorithm speed	efficiency
T3D 1 node	2756	1.00
T3D 32 nodes	109	0.79
T3D 64 nodes	61	0.71
T3D 128 nodes	41	0.53
iPSC Hypercube 8 nodes	1325	-
SGI cluster 1 node	2372	1.00
SGI cluster 6 nodes	416	0.95

Table 4
algorithm speeds in μsec/gp/ts on various machines.

efficiencies quoted should be regarded as pessimistic for any practical application of the code.

The algorithm speeds achieved are shown in table 4. These results illustrate several points. First, the T3D speeds are fast due to the large number of powerful nodes available. Respectable efficiencies are attained when it is taken into account that there is only a small amount of work on each processor for the present test case compared with potential applications. Careful examination of the performance of the code using the Cray Apprentice profiler shows that the cache use is inefficient and that a speed up by a factor of 2 is theoretically possible from reprogramming to avoid this problem. The Hypercube performance is included for reference to show that this machine is unsuited to this type of problem if only a small number of nodes are available.

The workstation cluster results show excellent performance and illustrate two important points. First, the small drop in parallel performance demonstrates that the networking combined with PVM is sufficiently fast to allow this type of cluster to perform as a powerful parallel machine for CFD calculations. The high efficiencies achieved also illustrate an important point about using the parallel algorithm effectively. For the cluster results the spanwise slices were solved completely on one node, thus limiting the communication during phase one to that of an *explicit start-up*. This division of the work was used due to the small number of processors available. To utilise the larger number of processors on the T3D it was necessary to divide each spanwise slice amongst several processors. This has the effect of reducing the overall parallel efficiency because of the communication costs incurred through solving each slice in parallel, although it should be remembered that the execution time is still decreased.

5. Conclusions

The parallel implementation of two and three-dimensional compressible and turbulent flow codes has been presented. Results obtained on the intel Hypercube, the Cray T3D and a Silicon Graphics workstation cluster have shown that these codes can make effective use of parallel computers.

There remains scope for improving the execution times of these codes. This would be

particularly valuable for the three-dimensional code on the T3D which will be used to tackle very large problems of practical interest. First, an effective transposition strategy needs to be investigated. The one used here is the simplest conceivable and does not take account of any optimised communication strategy. Since this part of the code degrades the parallel performance most there is significant scope for improvement here. Secondly, reprogramming to make better use of the data cache has the potential to improve execution times by a factor of two.

The most important conclusions of this work relate to the implications of the results for the application of these codes. The power of the T3D is required to complete a comprehensive study of the numerical characteristics of the three-dimensional code and then to study realistic industrial problems. The workstation cluster is sufficiently powerful to allow a fast enough turn around time for the study of unsteady aerofoil flows.

REFERENCES

1. K.J.Badcock, I.C.Glover, and B.E.Richards. Convergence acceleration for viscous aerofoil flows using an unfactored method. In *Second European conference on CFD*, pages 333–341. ECCOMAS, 1994.
2. K.J.Badcock and B.E.Richards. Implicit time stepping methods for the Navier-Stokes equations. In *12th AIAA CFD conference, San Diego.* AIAA, 1995.
3. K.J.Badcock and A.L.Gaitonde. An unfactored method with moving meshes for solution of the Navier-Stokes equations for flows about aerofoils. *submitted for publication, August,1994*, 1994.
4. K.J.Badcock and I.C.Glover. An implicit time stepping method for three-dimensional viscous flows. *submitted for publication, August 1994*, 1994.
5. T. Chyczewski, F. Marconi, R. Pelz, and E. Churchitser. Solution of the Euler and Navier-Stokes equations on a parallel processor using a transposed/Thomas ADIalgorithm. In *11th AIAA Computational Fluid Dynamics Conference.* AIAA, 1993.
6. P.J. Wesson. *Parallel Algorithms for Systems of Equations.* PhD thesis, Oxford University Computing Laboratory, Oxford, UK, 1992.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

FIESTA-HD: A Parallel Finite Element Program for Hydrodynamic Device Simulation

N. R. Aluru, K. H. Law, A. Raefsky and R. W. Dutton
Applied Electronics Laboratory, Stanford University, Stanford, CA 94305-4020.

The convective hydrodynamic transport model (HD) for semiconductor device simulation is studied by employing parallel and stabilized finite element methods. The HD model is shown to resemble the compressible Euler and Navier-Stokes equations and Galerkin/least-squares finite element methods, originally developed for compuational fluid dynamics equations, are extended to account for the strong nonlinear source terms present in the HD model. The complexity of the HD model demands enormous computational time. A parallel finite element device simulation program, FIESTA-HD, has been developed and run on distributed memory parallel computers including Intel's i860, Touchstone Delta and the IBM's SP-1 systems.

1. Introduction

Semiconductor device simulation has been based primarily on the drift-diffusion (DD) model for carrier transport, a simplification of the Boltzmann Transport Equation (BTE). With the scaling of silicon devices into deep submicron region, non-stationary phenomena such as velocity overshoot and carrier heating are becoming increasingly important to determine the characteristics of these devices. Due to the assumption of local equilibrium, the DD model cannot capture such non-stationary phenomena accurately. Although the direct solution of the BTE, for example via Monte Carlo method, can capture the above phenomena, the noise in the solution and the computational cost prevent it from wide usage for device simulation. An attractive alternative is to employ convective hydrodynamic or HD-like models. The convective hydrodynamic model can be directly derived from the zero, first and second moments of the BTE with a few simplifying assumptions [1]. These equations have a direct analogy to fluid dynamic equations. The development of stabilized and parallel finite element methods for the HD model is the subject of this paper.

Numerical simulation of the hydrodynamic semiconductor device model involves the solution of a coupled system of partial differential equations; namely the Poisson equation for the electrostatic potential and the electric field and the hydrodynamic equations for the electron and hole carriers describing the carrier concentration, velocities and temperature. The transport model is discretized employing stabilized Galerkin and Galerkin/least-squares finite element methods and the coupled equations are solved employing a fractional step solution strategy. In a fractional step solution strategy the Poisson, electron hydrodynamic and the hole hydrodynamic equations are solved iteratively in a decoupled manner until all sets of equations are solved to a given tolerance. Simulation

of the HD transport model on the present day workstations takes anywhere from a few minutes to several hours. The hydrodynamic device simulations are made practical by implementing the serial finite element program on distributed memory parallel computers such as Intel's i860, Touchstone Delta and the IBM's SP-1. The parallel finite element program is shown to produce results within reasonable time for engineering applications.

This paper is organized as follows: The hydrodynamic transport model is reviewed in Section 2. The development of a finite element formulation for the HD transport model is briefly presented in Section 3. The parallel computational model is summarized in Section 4 and numerical results are presented in Section 5. Section 6 summarizes the results of this work.

2. Hydrodynamic Model

The hydrodynamic transport model for semiconductor device simulation consists of the Poisson equation and nonlinear conservation laws for electrons and holes. The Poisson equation describes the conservation of charge in a semiconductor device and the nonlinear hydrodynamic equations for the electrons and holes describe the conservation of particle number, momentum and energy. Derived from the Maxwell's equations, the Poisson equation for computing the electrostatic potential and the electric field can be summarized as:

$$\nabla \bullet (\theta \nabla \psi) = \varepsilon(c_n - c_p - N_D^+ + N_A^-) \tag{1}$$

$$\mathbf{E} = -\nabla \psi \tag{2}$$

where ε, ψ, θ and $\mathbf{E}$ are the charge, the permittivity, the electrostatic potential and the electric field, respectively; c_n, c_p, N_D^+ and N_A^- are the concentrations of electrons, holes, ionized donors and ionized acceptors, respectively. The subscripts n and p denote the electron carrier and the hole carrier, respectively. The carrier concentrations are related to the electrostatic potential by the relations:

$$c_n = c_0 e^{\varepsilon(\psi - \varphi_n)/(k_b T_n)} \tag{3}$$

$$c_p = c_0 e^{\varepsilon(\varphi_n - \psi)/(k_b T_p)} \tag{4}$$

where c_0 is the intrinsic carrier concentration for the silicon material, k_b is the Boltzman constant, φ_n and φ_p are the quasi-fermi potentials for the carriers and T_n and T_p are the carrier temperatures.

The electron and hole hydrodynamic equations derived from the first three moments of the Boltzman Transport equation (BTE) are given as:

$$\frac{\partial c_\alpha}{\partial t} + \nabla \bullet (c_\alpha \mathbf{u}_\alpha) = \left[\frac{\partial c_\alpha}{\partial t}\right]_{col} \tag{5}$$

$$\frac{\partial \mathbf{p}_\alpha}{\partial t} + \mathbf{u}_\alpha(\nabla \bullet \mathbf{p}_\alpha) + (\mathbf{p}_\alpha \bullet \nabla)\mathbf{u}_\alpha = (-1)^j \varepsilon c_\alpha \mathbf{E} - \nabla(c_\alpha k_b T_\alpha) + \left[\frac{\partial \mathbf{p}_\alpha}{\partial t}\right]_{col} \tag{6}$$

$$\frac{\partial w_\alpha}{\partial t} + \nabla \bullet (\mathbf{u}_\alpha w_\alpha) = (-1)^j \varepsilon c_\alpha (\mathbf{u}_\alpha \bullet \mathbf{E}) - \nabla(\mathbf{u}_\alpha c_\alpha k_b T_\alpha) - \nabla \bullet \mathbf{q}_\alpha + \left[\frac{\partial w_\alpha}{\partial t}\right]_{col} \tag{7}$$

where $\mathbf{u}_\alpha$, $\mathbf{p}_\alpha$, w_α and $\mathbf{q}_\alpha$ are the velocity vector, momentum density vector, energy density and heat flux vector of the carrier α. (For electron, $\alpha = n$ and $j = 1$; for holes, $\alpha = p$ and $j = 2$.) The terms $[\ ]_{col}$ represents the rate of changes in the particle concentration, momentum and energy due to the collision of carriers; the collision terms can be approximated by their respective relaxation times and the expressions can be found in [1]. The hydrodynamic system as stated above contains fewer equations than unknowns. In order to facilitate a solution, the following constitutive approximations are introduced for the momentum and energy density:

$$\mathbf{p}_\alpha = m_\alpha c_\alpha \mathbf{u}_\alpha \tag{8}$$

$$w_\alpha = \frac{3}{2} c_\alpha k_b T_\alpha + \frac{1}{2} m_\alpha c_\alpha |\mathbf{u}_\alpha|^2 \tag{9}$$

where m_α is the mass density of the carrier and k_b is the Boltzmann constant.

Similar to the HD equations, the Euler and Navier-Stokes fluid equations can be physically interpreted as the conservation of particle, momentum and energy. However, the HD equations are not identical to either the Euler equations or the Navier-Stokes equations. While the HD equations do not contain the viscous terms, they are not the same as the Euler equations because of the presence of the heat conduction term in the energy equation. Furthermore, the highly nonlinear source terms in the HD model are absent in the fluid models. It can be shown that the HD system resembles the flow of a real compressible fluid given by the Euler equations, in the presence of electric field and with the addition of a heat conduction term and the highly nonlinear source terms [1].

3. Finite Element Formulation

For the elliptic Poisson equation, a standard Galerkin finite element method has been employed for the numerical solution. However, the standard Galerkin finite element method is known to exhibit spurious oscillations for the advective-diffusive type equations like the HD equations when the physical diffusion present in the system is small. In this work, we employ the Galerkin/Least Square (GLS) method [3] and enhance it to account for the strong nonlinear source terms of the HD device equations. The temporal behavior of the HD equations is discretized using a discontinuous Galerkin method in time [4]. The basic formulation of the space-time GLS discretization scheme can be summarized as follows:

1. First, the weak form of the given partial differential equation (the strong form) is obtained by multiplying the strong form with an arbitrary test function. We then integrate the resulting system by parts. It can be shown that the strong form and the weak form are equivalent and the solution to the weak form is also the solution to the strong form (i.e. the governing partial differential equations).

2. A least-squares term of a residual type is introduced to the weak form of the given partial differential equation so that the numerical stability of the system is enhanced.

Furthermore, a discontinuity-capturing term is added to overcome the undershoot and overshoot phenomena. The least-squares and discontinuity capturing terms vanish when the exact solution is substituted to the weak form, thus establishing the consistency of the method.

3. The trial and test functions used for the nonlinear FEM equations are taken to be a linear combination of linear basis functions.

4. The nonlinear system is solved using a Newton iterative scheme by linearizing the nonlinear equations with respect to the unknown trial solution.

A comprehensive discussion on the development of the finite element space-time GLS formulation of the HD semiconductor device equations is given in [1]. The Galerkin method for the Poisson equation and the space-time Galerkin/least-squares method for the electron and hole hydrodynamic equations are shown to be stable and consistent. In the case of the electron and hole conservation laws, a Clausius-Duhem inequality, also defined as the basic stability requirement for nonlinear conservation laws, is derived and the space-time GLS method is shown to satisfy this inequality. Hence, the numerical methods employed in this work are also known as stabilized finite element methods.

A fractional-step/staggered scheme is applied to solve the coupled systems. The Poisson equation is first solved for the electrostatic potential and the electric field. The computed electric field values are used in the HD equations to solve for the concentrations, velocities and temperature. The concentrations obtained from the HD equations provide a new source term to the Poisson equation. This staggered procedure of alternatively solving the Poisson and HD equations is repeated until both the equations are solved to a desirable tolerance.

4. Parallel Computational Model

The single-program-multiple-data (SPMD) paradigm has emerged as a standard model to create parallel programs for engineering applications on distributed memory parallel computers [2]. In this approach, problems are decomposed using some well known domain decomposition techniques. Each processor of the parallel machine solves a partitioned domain. Data communication between domain partitions are performed among processors through message passing.

For a large scale engineering software, besides optimizing the parallel kernels for linear algebraic and/or matrix computations, attention must be paid to the overall program structure and the data flow among the program modules. A typical finite element program consists of the following tasks: pre-processing, element generation, matrix formation, solution of a system of linear equations and post-processing. The pre-processor supports problem definition, grid generation, I/0 and other file management functions. Generally, the pre-processing routines take negligible time and are inherently serial. The parallelization is thus focused primarily on the numerical PDE solvers. In FIESTA-HD, the linear equation solvers currently employed are the GMRES method for solving the non-symmetric linear equations of the HD systems and the conjugate gradient method for the symmetric linear equation for the Poisson system. For a finite element program with iterative solvers, the interprocessor communication is limited primarily to the linear solver.

Special care, however, is needed to set up the data structure required by each processor and to ensure proper data flow between the pre-processor and the parallel PDE solvers. The parallel program organization of FIESTA-HD is depicted as shown in Figure 1.

Initial development of the parallel FIESTA-HD program took place on a 32-node Intel i860 computer. The code has since been ported to the Intel Touchstone Delta and the IBM SP1 computers. For the Intel-based implementation, a front end workstation is used for the pre-processing tasks. For the IBM SP1 parallel computer, the pre-processor resides on a master node (which also serves as a slave processor for the parallel PDE solvers) and a more efficient model is implemented, taking advantage of the memory available on the SP1. The porting of the code from the i860 to the Delta and to the SP1 takes less than a week. For each case, majority of the work has been to re-structure the pre-processing module.

5. Numerical Results

To demonstrate the utility of FIESTA-HD, we have run simulations using increasingly large and complex realistic device structures on the parallel computers. The results are summarized in Figures 2-4. Figures 2 and 3 show, respectively, the simulation results of a bipolar transistor on the Delta machine and a 2-D two carrier p-n diode on both the i860 and the IBM SP1. The results clearly show the portability and scalability of FIESTA-HD simulator on various parallel computers. Figure 4 shows a comparison of the CPU times of FIESTA-HD on the IBM RS/6000 workstation, the i860 (32 processors), and the IBM SP-1 (8 processors) for several different diodes and bipolar transistors. As grids scaled to modest and large sizes, the parallel codes performed significantly better than the workstation version. We routinely achieved more than an order-of-magnitude reduction in execution time. Moreover, using these parallel machines, we have been able to solve very large device structures for which a serial solution could not be obtained due to resource constraints.

6. Summary

In this note, we have briefly discussed the hydrodynamic model for semiconductor device simulation and the resemblance of the HD device equations with the Euler and Navier-Stokes fluid equations. A space-time Galerkin/Least-Squares finite element method is proposed for the solution of the HD equations. A SPMD programming model is used in the parallel implementation of the device simulator, FIESTA-HD. Our experience has clearly demonstrated the portability of FIESTA-HD on distributed memory parallel computers. Numerical results are shown to demonstrate the robustness and accuracy of the numerical schemes. Taking advantage of the advances in parallel computers with stable numerical schemes, we are able to perform simulations with more complex and realistic device models.

7. Acknowledgment

This research was sponsored by ARPA, contract No. DAAL 03-91-C-0043.

REFERENCES

1. N. R. Aluru, *Parallel and Stabilized Finite Element Methods for the Hydrodynamic Transport Model of Semiconductor Devices,* Ph.D Thesis, Department of Civil Engineering, Stanford University, June, 1995.
2. B.P. Herndon, N.R. Aluru, A. Raefsky, R.J.G. Goossens, K.H. Law and R.W. Dutton, "A Methodology for Parallelizing PDE Solvers: Application to Semiconductor Device Simulation," Seventh SIAM Conference on Parallel Computing, San Francisco, CA, 1995.
3. F. Shakib, *Finite Element Analysis of the Compressible Euler and Navier-Stokes Equations,* Ph.D. Thesis, Department of Mechanical Engineering, Stanford University, November, 1988.
4. C. Johnson, U. Navert and J. Pitkaranta, "Finite Element Methods for Linear Hyperbolic Problems," *Computer Methods in Applied Mechanics and Engineering,* 45:285-312, 1984.

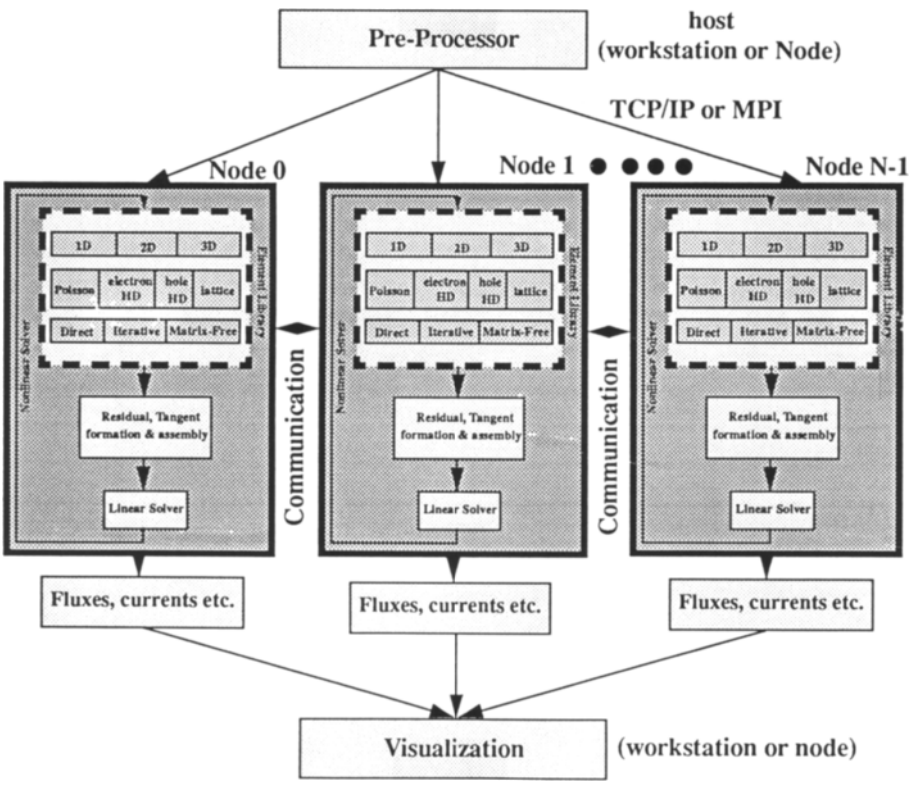


Figure 1. Program Organization of Parallel HD Device Simulator, FIESTA-HD

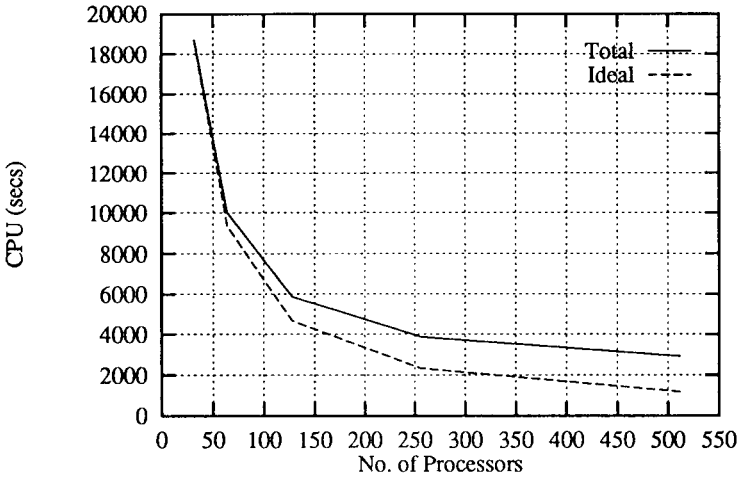


Figure 2. Total CPU time on the Touchstone Delta machine for a bipolar transistor

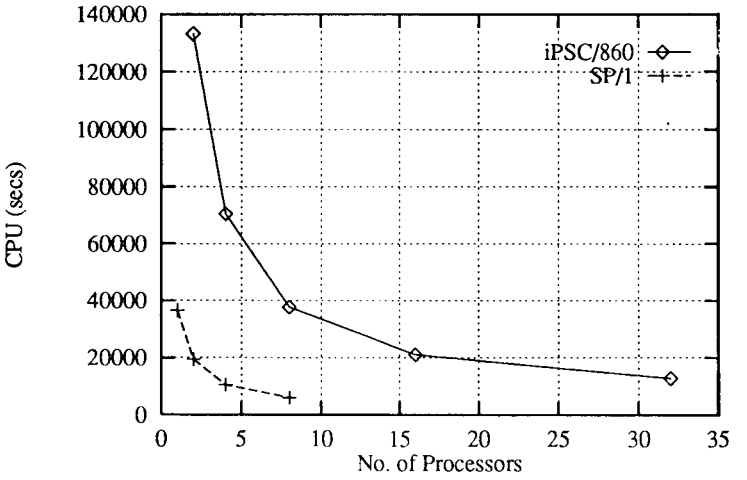


Figure 3. CPU time comparison on i860 and SP-1 for the 2D pn silicon diode example

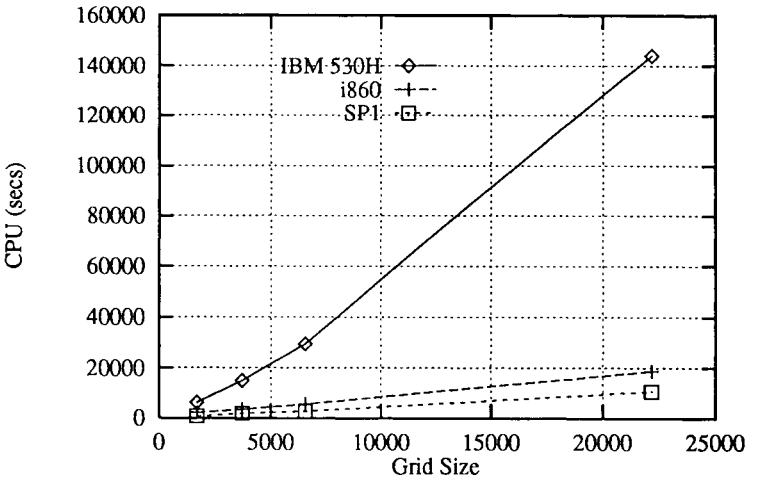


Figure 4. Execution Times of FIESTA-HD on serial and Parallel Computers

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

Parallel computation of a tip vortex induced by a large aircraft wing

Ryozo Ito[a] and Susumu Takanashi[b]

[a]Space Systems Department, Systems Engineering Group, DAIKO DENSHI TSUSHIN Ltd., 2-1 Ageba-cho, Shinjuku-ku, Tokyo 162, Japan
E-mail address: rito@nal.go.jp

[b]Computational Aerodynamics Section, National Aerospace Laboratory, 7-44-1 Jindaiji-higashi, Chofu, Tokyo 182, Japan

Three dimensional Navier-Stokes simulations about the wing of the Boeing 747 have been carried out using a parallel vector computor called "NWT". The main objective of this study is to simulate the wing-tip vortex.

The governing equations are 3-D Reynolds-averaged thin-layer Navier-Stokes equations which are discretized by finite volume method with a TVD upwind scheme. The domain around the wing is decomposed into 24 subdomains, but the grid system is basically in C-O topology. The method of the parallelization is to share the load of computation for each subdomain to each PE. Computation using a grid with 11-million grid-points shows an existence of a strong tip vortex induced by the wing.

1. INTRODUCTION

The strong trailing vortices induced by a large aircraft like the Boeing 747 are threat for the following aircrafts. Actually, many incidents caused by such vortical flows have been reported so far[1]. In order to keep air-trafic safety, it is important to understand the characteristics of the wake vortices by means of both the experimental and the numerical studies.

In the present study, as a preliminary case, 3-D Navier-Stokes simulations using finite volume method have been carried out for the wing of the Boeing 747-200. Computations have been done by using parallel vector computor called "NWT" at the National Aerospace Laboratory, Japan.

2. NUMERICAL METHOD

Governing equations are the Reynolds averaged thin-layer 3-D Navier-Stokes equations. These equations are discretized by the finite volume method. The time-integration algorithm of these equations is the implicit approximate factorization method with locally varying time stepping. The convective terms are formulated by Chakravarthy's 3rd order TVD scheme[2]. Details of these processes are described in Ref.3. Turbulent eddy viscosity is

evaluated by the Baldwin-Lomax algebraic model[4]. Besides, this flow solver is applicable for a multi-domain grid.

3. COMPUTATIONAL GRID

The simple C-O type of body-fitted grid system is used. Figure 1 shows the upper-half region of the grid. The grid around the wing with wake is algebraically generated using transfinite interpolation[5]. The total number of grid-points is 10,802,017(577 in streamwise direction, 97 in spanwise direction, and 193 in normal direction). The minimum spacing normal to the wall is 3.0×10^{-5}, and the far field boundary is located 7.0 root chord length away from the root of the wing. Outflow boundary is located 4.1 root chord length downstream from the trailing edge of the wing-tip.

The domain of the whole grid is equally decomposed into 24 subdomains. First, the whole grid is decomposed equally into three subdomains, including the wing-surface, the upper side of wake and the lower side of wake. Next, each of these three subdomains is decomposed equally into eight subdomins in spanwise direction, then the total number of the subdomains is 24. Besides, in order to estimate the dependence of the numerical solution to the grid size, the coarse grid has been made by using every second points in the original grid.

4. PARALLEL IMPLEMENTATION

4.1. NWT

The specifications of the NWT as a parallel computor are as follows. NWT means "Numerical Wind Tunnel".

- Parallel vector computor with distributed memory architecture.
- Up to 140 PEs(Processor Elements) can work parallelly.
- Each PE is a vector computor whose performance is about 1.7 GigaFLOPS.
- User's memory for each PE is about 200 MegaBytes.
- Each PE communicates each other through the cross-bar network.
- *Intra*-PE data transfer rate : 6.4 GigaBytes/sec
- *Inter*-PE data transfer rate : 0.4 GigaBytes/sec

Inter-PE data transfer rate is much less than *intra*-PE data transfer rate. So, minimization of the *inter*-PE data transfer is important for obtaining high performance of this computor.

4.2. METHOD OF PARALLELIZATION

Parallel implementation has been done by means of sharing the load of computation for each subdomain to each PE. Number of PEs used should be less than or equal to the number of subdomains. This method is so straightforward that the reconstruction of the code for the parallel implementation can be easily accomplished. However, to calculate the values of the numerical fluxes on the cells along the edge of a subdomain, their counterparts of the neighboring subdomain are required. This causes *inter*-PE data transfer among PEs every one iteration. Figure 2 illustrates this aspect in case that there are only two subdomains.

5. RESULTS

Computations have been done on NWT under the following flow conditions. Free stream Mach number is 0.8, angle of attack is 6.72 degrees, and Reynolds number based on the root chord length of the wing is 5.76×10^6. Number of PEs used is 24 in all cases except for the case of testing the parallel efficiency (section 6.3.).

5.1. Validation Study

The numerical solutions obtained here have been compared with the wind tunnel test data for the CFD code validation. The wind tunnel test was done by the Boeing Commercial Airplane Company[6] using the transonic 11-foot pressure tunnel at NASA Ames Reseach Center. Figures 3 show the comparison of the pressure distributions on the wing with the wind tunnel test data. The agreement between the computations and the experimental data becomes better as the location of the span-station becomes closer to the wing-tip. This tendency is considered to be consistent with the fact that the experiment was done for the wing-*body* configuration but not for the wing alone. Both the result of the fine grid and the result of the coarse grid show a good agreement with the experimental data on the wing-tip. The number of grid-points is considered to be sufficient for computing the pressure distribution on the wing surface in either case.

5.2. Wing-tip vortex

Figures 4 show the front view of the computed stream lines shedding from the wing-tip overlapping with the computational grids. Stream lines form a vortex with counterclockwise rotation in either case. The vortex captured in the coarse grid gets looth quickly, while the result of the fine grid shows that the the size of the vortex in the cross-flow direction is constant. From this point of view, the number of grid-points in the fine grid is considered to be enough to capture the wing-tip vortex because the size of the vortex in the cross flow direction does not depend on the grid-point density. Figures 5 show the top view and the side view of the stream lines mentioned above.

5.3. Speed-up

In case of the fine grid, the computation done by each PE took about 10 hours to make 10,000 iterations. The CPU-time for *inter*-PE data transfer in this case is about 6 minutes, i.e. only about 1% of the whole computation. The parallel efficiency can not be evaluated with the fine grid because the number of grid-points is too large to execute the computation within one PE, but the CPU-time measured here implies that the parallel efficiency is about 0.99. The CPU-time for *inter*-PE data transfer mentioned above means the sum of the CPU-time for generating a packet to send and the CPU-time for sending the packet itself. The former does not depend on the grid size, but the latter is proportional to the grid size.

Besides, in order to evaluate the parallel efficiency directly, a grid which consists of only 177,625 points has been prepared. This grid has been decomposed equally into 24 sub-domains also. In this case, NWT using 24 PEs executes the computation 22.3 times faster than NWT using one PE, hence the parallel efficiency is 0.93. As the number of grid-points becomes larger, the CPU-time for the *inter*-PE data transfer is considered to become relatively shorter. So, the parallel efficiency in case of the fine grid would be expected to achieve 0.93 at least if the computation could be done within one PE. Figure 6 shows the

relationship between the number of PEs and the CPU-time for 200 iterations. Speed of computing is almost exactly proportinal to the number of PEs utilized in these computations.

6. CONCLUDING REMARKS

Navier-Stokes simulations for the wing-tip vortex induced by the wing of the Boeing 747-200 have been carried out by using a parallel vector computor. The solver for the multi-domain grid has been highly parallelized by sharing the load of computation in each subdomain equally to each PE.

As a future work, the vortex simulation should be attempted at the flow condition of lower speed and higher angle of attack as the aircraft is in take-off or landing.

REFERENCES

1. T. Lee and C.E. Lan, AIAA Paper, 94-1882-CP (1994).
2. S.R. Chakravarthy, AIAA Paper, 86-0243 (1986).
3. M. Tachibana and S. Takanashi, NAL SP-10, (1989) (in Japanese).
4. B.S. Baldwin and H. Lomax, AIAA Paper, 78-257 (1978).
5. L.E. Eriksson, AIAA Journal, Vol.20, No.10 (1982).
6. E.N. Tinoco, private communication, 1989.

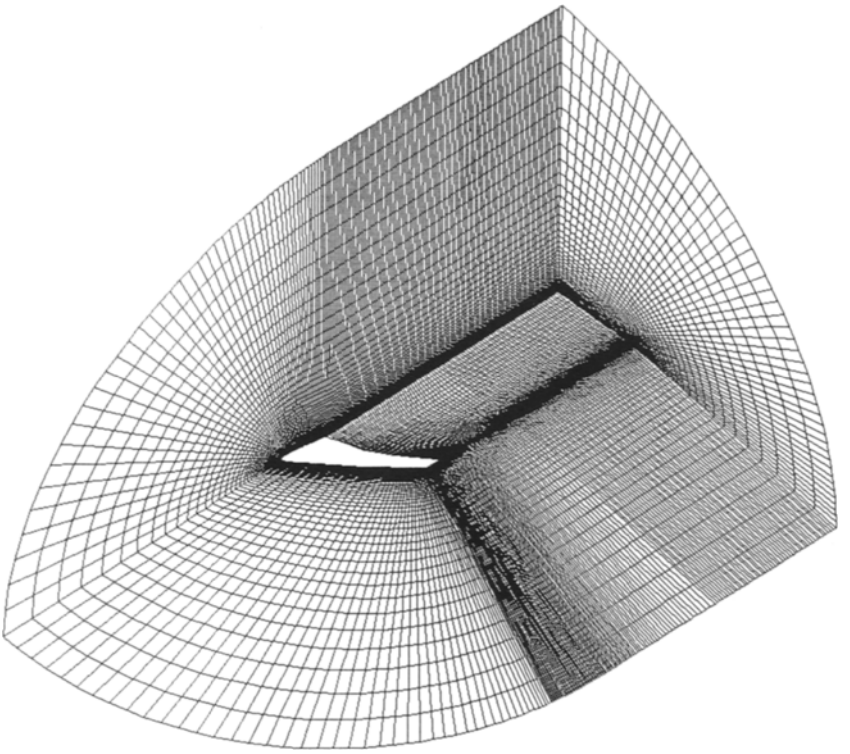


Figure 1. Upper-half region of the grid.

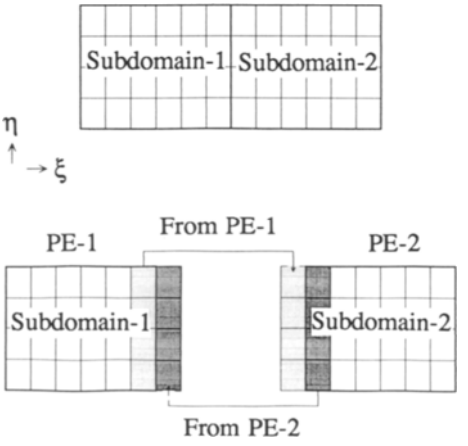
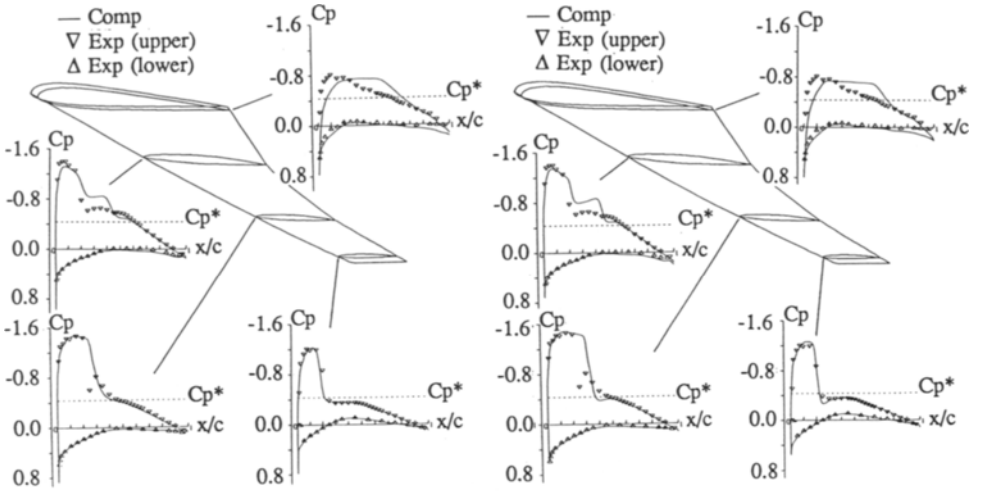
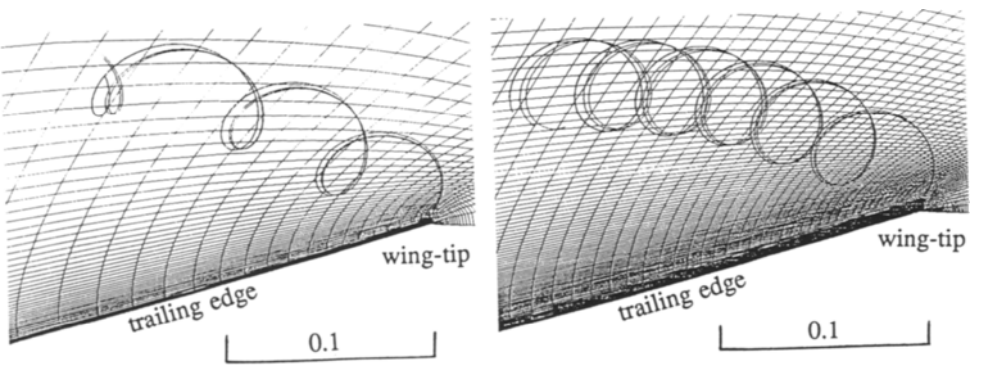


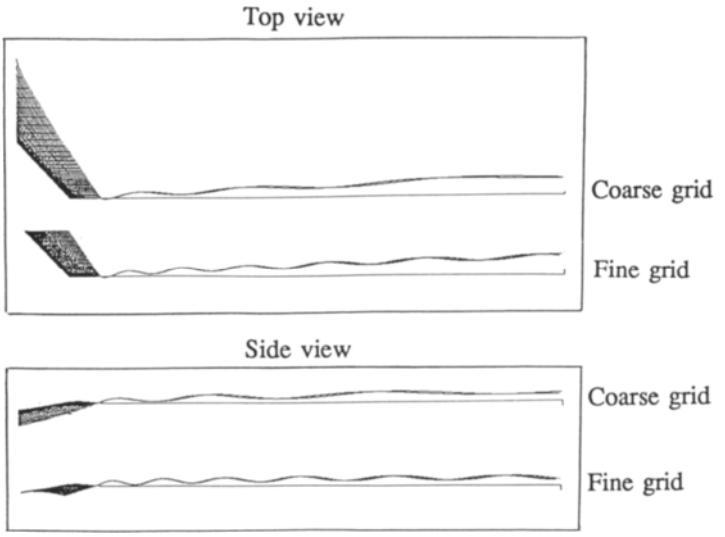
Figure 2. *Inter*-PE data transfers caused by the domain decomposition.



Figures 3. Comparison of the Cp distributions on the wing with the experimental data.



Figures 4. Front view of the computed stream lines shedding from the wing-tip.



Figures 5. Top view and side view of the computed stream lines shedding from the wing-tip.

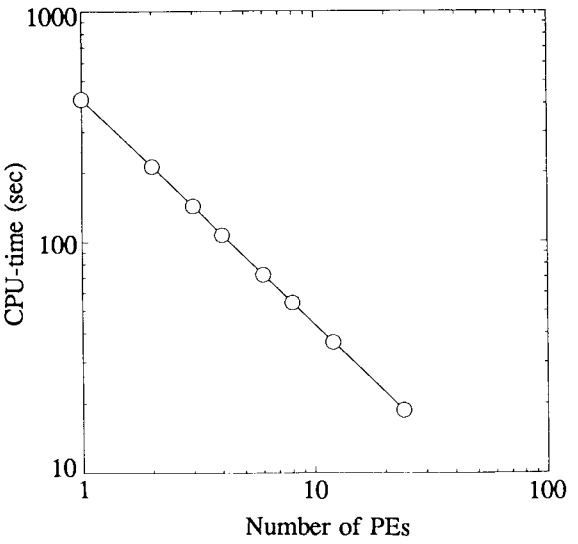


Figure 6. Speedup of the computation.

This Page Intentionally Left Blank

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Shape design optimization in 2D aerodynamics using Genetic Algorithms on parallel computers

Raino A.E. Mäkinen[a], Jacques Periaux[b] and Jari Toivanen[a]

[a]University of Jyväskylä, Department of Mathematics, P.O. Box 35, FIN-40351 Jyväskylä, Finland

[b]Dassault Aviation, Aerodynamics and Scientific Strategy, 78 Quai Marcel Dassault, 92214 St Cloud Cedex, France

Two shape optimization problems for two-dimensional airfoil design are presented. The first one is a reconstruction problem for an airfoil when the velocity of the flow is known on the surface of airfoil. The second problem is to minimize the shock drag of an airfoil at transonic regime. The flow is modeled by the full potential equations. The discretization of the state equation is done using the finite element method and the resulting non-linear system of equations is solved by using a multigrid method. The non-linear minimization process corresponding to the shape optimization problems are solved by a parallel implementation of a Genetic Algorithm. Finally, numerical experiments are computed on a SP2 parallel computer. The results from the experiments are compared with those obtained using a gradient based minimization method.

1. Introduction

Numerical shape optimization has been under extensive study during the past twenty years [5], [6], [8]. In aerodynamics a typical problem is the shape design optimization of two-dimensional airfoil. In these problems the aim – usually denoted by the cost function – can be for example to minimize the drag or to maximize the lift. Another possibility is to consider an inverse problem. For example find an airfoil shape of airfoil such that the drag caused by shock is minimized or find the airfoil shape when the velocity or the pressure on the surface of airfoil are given. A inviscid flow analysis model of this kind of problems is usually represented by the full potential or Euler equations. The finite element method is a common approximation to discretize the state equation describing the flow. The multigrid methods give an efficient way to solve the state equation [3].

A traditional way originating from control theory [5], [8] to solve a shape optimization problem is to use a gradient method as a deterministic optimizer. In this case it is necessary to perform a sensitivity analysis, i.e. to compute the gradient of the cost function. The regularity requirements for gradient methods are quite strict. Usually the cost function must be at least a smooth function. Another problem encountered by conventional optimization method is a possible encountered by conventional optimization method convergence to a local minimum in the case of a multimodal cost function. Therefore global

optimization methods should be used when the cost function is not unimodal. Gradient methods are basically sequential methods whose parallel implementation is not obvious.

Recently, there has been increasing interest to use evolutive optimization methods such as Genetic Algorithms (GAs) in shape design optimization. GAs provide several advantages: (i) they are global optimization methods based on fitness function evaluations only; hence the gradient information is not needed and the cost function can be a multimodal function. (ii) the cost function can be nonsmooth or even discontinuous. On the other hand GAs work with evolutive populations generated stochastic selection, crossover and mutation operators. This non pointwise property render parallel implementations of GAs quite naturally. By using a master-slave prototype, the communication requirement of parallelized GAs is very small (reduced to design variables and cost function values) Therefore a cluster of workstations can be efficiently used to carry out the optimization.

In the first three sections the transonic potential flow, the reconstruction problem and the shock reduction problem are introduced briefly. Then GAs and their parallel implementation are considered. In the next section some numerical experiments are performed in IBM SP2 parallel computer. In the end some conclusions and perspective are made.

2. Transonic potential flow

The transonic potential flow equation with associated boundary conditions for nonlifting airfoil is

$$\begin{aligned} -\nabla(\rho\nabla\Phi) &= 0 \quad \text{in } \Omega \\ \rho\frac{\partial\Phi}{\partial n} &= q \quad \text{on } \partial\Omega, \end{aligned} \tag{1}$$

where Φ is the normalized velocity potential and ρ is the density. The formula for the density ρ is given for example in [3]. The half of computational domain Ω around a symmetric airfoil is shown in Figure 1.

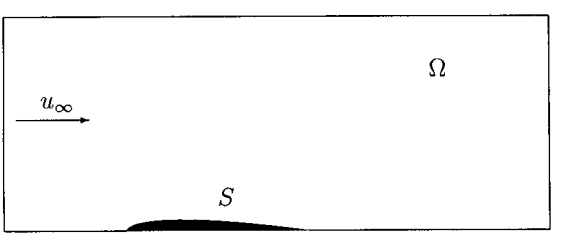


Figure 1. Problem geometry.

The state equation (1) is discretized with the finite element method. In supersonic region the artificial viscosity is included by replacing the density ρ with an artificial density, which is obtained by adding an upwind derivative term to the density. Due to this artificial viscosity the physical shock can be captured. Then state equation (1) is solved by a SLOR-multigrid method [3].

3. Reconstruction problem

In the shape reconstruction problem the aim is to find the shape of a target airfoil when the velocity is known on the surface S. This problem can be formulated as a minimization problem

$$\min_{\alpha \in U_{ad}} J_1(|\nabla \Phi(\alpha)|), \tag{2}$$

where α is the vector containing the design variables defining the shape of an airfoil and U_{ad} is the set of admissible designs. The cost function is

$$J_1(|\lambda|) = \int_0^L (|\lambda| - |\lambda_t|)^2 \, dx, \tag{3}$$

where λ_t is the target velocity obtained by using the target airfoil and L denotes the chord of the airfoil. This reconstruction problem has been considered for example in [7].

4. Shock reduction problem

The shape optimization problem corresponding to the shock drag reduction is formulated as a minimization problem

$$\min_{\alpha \in U_{ad}} J_2(|\nabla \Phi(\alpha)|), \tag{4}$$

where the notations are the same as in (2). The cost function is

$$J_2(|\lambda|) = \int_0^L \left(\min \left(\frac{\partial |\lambda|}{\partial x}, 0 \right) \right)^2 dx. \tag{5}$$

In order to obtain a feasible design, the airfoils in the set U_{ad} must have an area greater than a given constant. This can be added to the optimization problem as a geometrical constraint. This shock reduction problem have been considered for example in [2] and [7].

5. Parallelization of Genetic Algorithm

Gradient algorithms for solving non-linear optimization problems can be efficient, but they may have several drawbacks. Some of them are: (i) possible convergence to a local minimum, (ii) strict regularity requirements for the cost function, (iii) need gradient information, (iv) they are sequential algorithms. Therefore, the parallelization must be done within one function evaluation in these pointwise gradient algorithms.

Recently, there has been increasing interest in global optimization methods based on Genetic Algorithms (GAs). These algorithms have been developed to simulate Darwin's principle of *survival of the fittest* [4]. The GAs are inherently parallel in the sense that they work with populations. After a generation is formed, the fitness function (cost function in the classical terminology) values corresponding to the individuals of the populations can be computed simultaneously.

In the particular cases when the fitness function evaluations are much more costly than the genetic mechanism, it is highly recommended for cost-efficiency to parallelize the evaluation of fitness function within each generation. A simple way to do this is

to use the master-slave prototype as shown in Figure 2. In this approach an optimal load balancing is obtained when the generation size is a multiple of the number of slave processes and each fitness function evaluation requires the same amount of time.

For optimal shape design application considered in this paper, the genetic terminology is the following: (a) the population is a set of airfoils, (b) the genes are the design parameters defining the airfoil shape and (c) the fitness function is the cost function of the design.

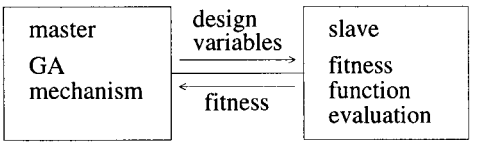


Figure 2. Master and one slave in parallel implementation of GAs.

6. Numerical examples

The parametrization of the airfoil shape is defined using a Bezier curve [1] with nine control points. The design variables are the seven y-coordinates of the inner Bezier control points. The control points on leading and trailing edge remain fixed during the optimization process. We have box constraints, i.e. upper and lower limits, for the search space of the design variables.

In all numerical experiments the airfoils are operating at transonic mach number with shocks ($M_\infty = 0.8$.) The discretization is done using 80×24 bilinear elements (41 nodes on the airfoil). During the optimization process a isotopology moving mesh generator is used, i.e. the nodal coordinates are smooth functions with respect to the design variables.

The GAs used in the experiments are founded on the simple Genetic Algorithms introduced in [4]. The main difference consist of floating point coding, tournament selection and elitism strategy. Only one crossover site is consider and the mutation done using a special distribution promoting small mutations. The parameters of GAs are shown in Table 1. From these parameters it is possible to come to a conclusion that it is necessary to compute 1601 fitness function values during one optimization run using the GAs.

Table 1
The parameters in GAs.

Population size	17
Elitism	1
Generations	100
Tournament Size	3
Crossover probability	0.85
Mutation probability	0.2

All experiments are also performed for comparison using the sequential quadratic programming (SQP) algorithm from the NAG Fortran library from Numerical Algorithms Group Ltd, Oxford. This method requires the computation of gradient. The sensitivity analysis is done according to [7]. The artificial viscosity in flow analysis solver is chosen in such a way that we obtain a smooth artificial density. Since we use a moving mesh, the exact gradient and a smooth artificial density the SQP algorithms should provide very good results.

First test case: inverse problem ($M_\infty = 0.8,\ \alpha = 0°$). We considered the reconstruction of the NACA0012 airfoil. The initial population in the GAs is chosen randomly. The initial design for the SQP is the parabola going through the points (0,0), (0.5,0.06) and (1,0). The results are shown in Figures 3, 4 and 5 and in Table 2. In the figures and tables 'GAs' and 'SQP' correspond to the final results obtained using respectively the GA and SQP methods and 'NACA' corresponds to the NACA0012.

Table 2
The cost function values in the numerical examples.

profile	reconstruction	shock reduction
GAs	1.58×10^{-5}	0.2581
SQP	3.14×10^{-7}	0.2506
NACA	0	3.6436

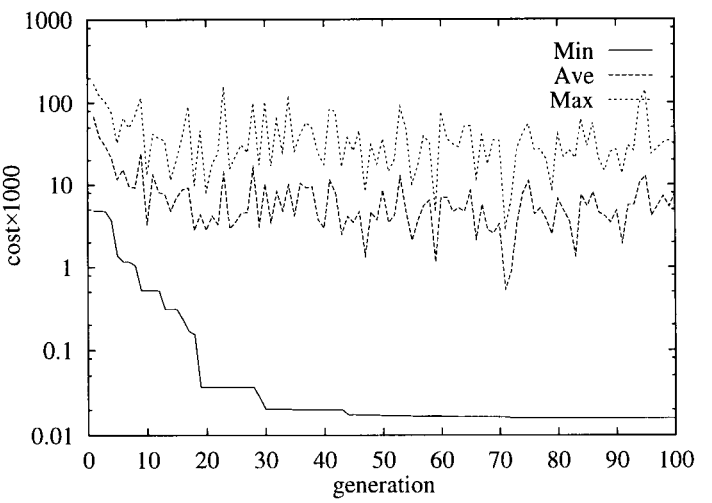


Figure 3. The evolution of population in the reconstruction problem.

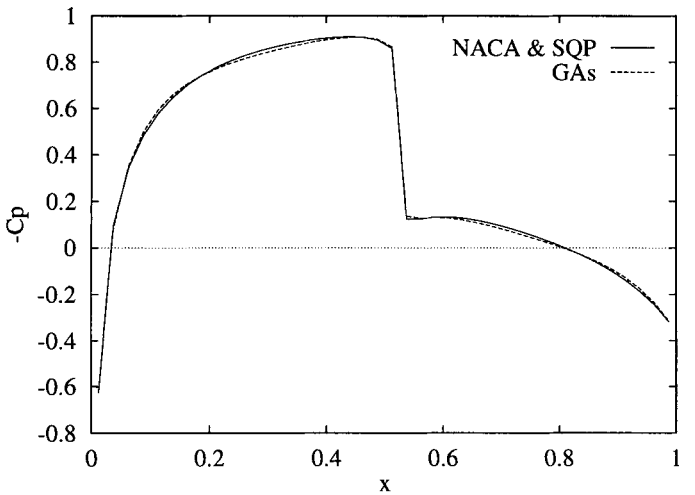


Figure 4. The surfacic pressure coefficients in the reconstruction problem.

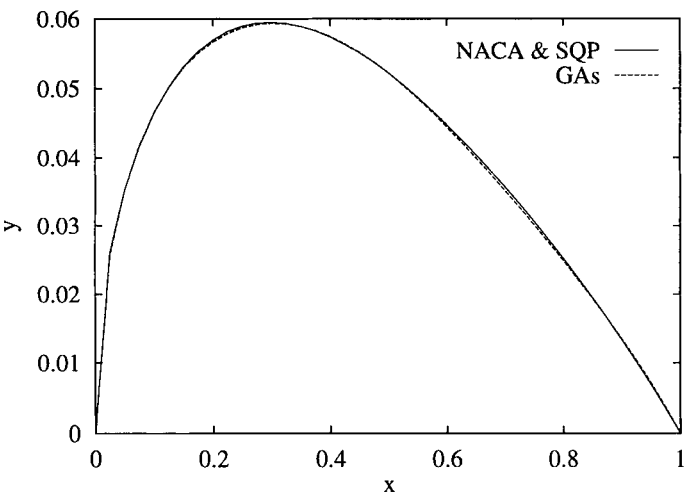


Figure 5. The profiles in the reconstruction problem.

Second test case: optimization problem ($M_\infty = 0.8$, $\alpha = 0°$, constant area). In this shock reduction problem the airfoil is constrained to have the same given area as the NACA0012 airfoil. With the SQP the NACA0012 airfoil is the initial design. The results are shown in Figures 6 and 7 and in Table 2. It can be observed that the initial shock has disappeared after the optimization process.

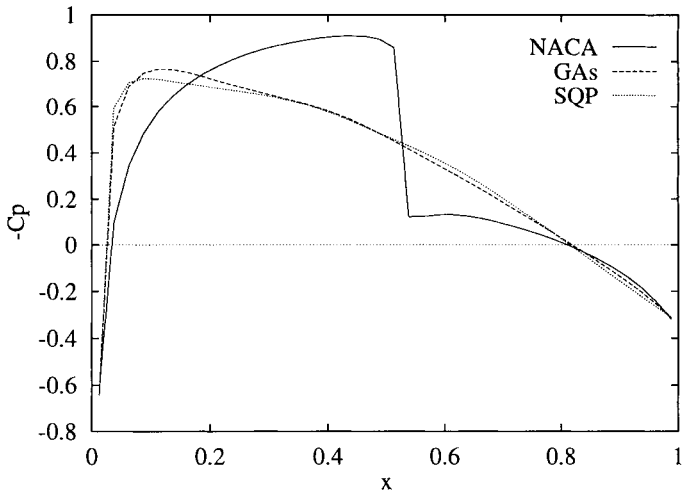


Figure 6. The surfacic pressure coefficients in the shock reduction problem.

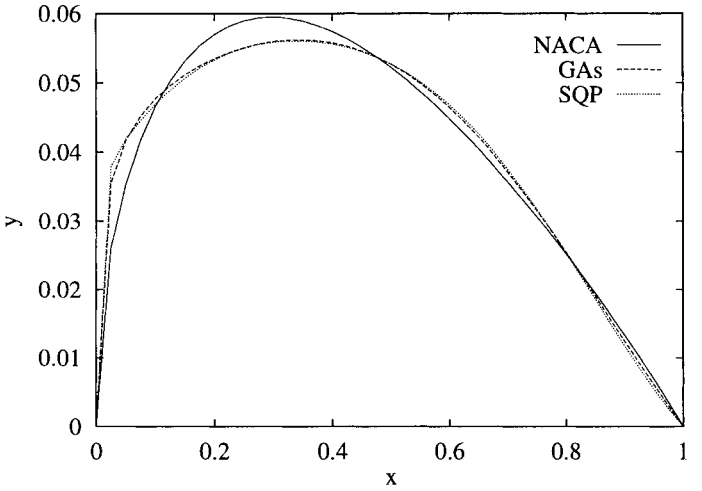


Figure 7. The profiles in the shock reduction problem.

The efficiency of the parallelized GAs is studied in IBM SP2 parallel computer. The message passing was done utilizing the MPI standard. The results in the shock reduction problem are shown in Table 3. A good scalability can be observed from Table 3.

Table 3
The elapsed times in seconds and speedups for the shock reduction problem.

slaves	time	speedup
1	18588	1
2	9378	1.98
4	4758	3.91
8	2432	7.64

The SQP required 22 function and gradient evaluations in 20 iterations in the shock reduction problem. The elapsed time was 315 seconds. Despite the parallelization of GAs there is still an important CPU gap between the deterministic and Genetic Algorithms. For efficiency and accuracy purposes hybridization of both approaches (the best airfoil from GAs is used as an initial guess for SQP) is clearly the next step to be investigated.

7. Conclusions

The designs obtained using the GAs are close to the optimal designs. The speedups of parallelized GAs are very promising. In these numerical examples the SQP was more accurate and efficient since the problems were quite simple and well tailor made. In the future GAs should be tested with more difficult problems such as transonic multi-point design problems in which gradient based methods are not expected to work so well.

REFERENCES

1. R.H. Bartels, J.C. Beatty and B.A. Barsky, An Introduction to Splines for use in Computer Graphics and Geometric Modelling, Morgan Kaufmann, Los Altos, 1987.
2. V. Danek and R. Mäkinen, Optimal design for transonic flows, International Series of Numerical Mathematics, **99**, 129–136, 1991.
3. H. Deconinck and C. Hirsch, A multigrid method for the transonic full potential equation discretized with finite elements on an arbitrary body fitted mesh, J. Comp. Phys., **48**, 344–365, 1982.
4. D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, Massachusetts, 1989.
5. J. Haslinger and P. Neittaanmäki, Finite Element Approximation for Optimal Shape Design, John Wiley & Sons, Chichester, 1988.
6. A. Jameson, Aerodynamic design via control theory, J. Sci. Comput., **3**, 233–260, 1988.
7. R.A.E. Mäkinen, Optimal shape design for transonic potential flows, Finite elements in fluids, K. Morgan, E. Oñate, J. Periaux, O.C. Zienkiewicz (Eds.), CIMNE/Pineridge press, 457–466, 1993.
8. O. Pironneau, Optimal Shape Design for Elliptic Systems, Springer-Verlag, New York, 1984.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

A model for performance of a block-structured Navier-Stokes solver on a cluster of workstations

M.E.S. Vogels[a]

[a]National Aerospace Laboratory NLR, P.O. Box 90502, 1006 BM Amsterdam, The Netherlands, e-mail: vogels@nlr.nl

To estimate the breakdown of parallel performance of a multi-block Navier-Stokes solver on a cluster of workstations, and to find the determining elements for this performance result, the performance has been modelled. The performance model is specific for the laminar flow over a flat plate, and uses network characteristics of a cluster of 3 IBM RISC workstations, interconnected through ethernet with PVM as message passing system.

The performance model predictions are in good agreement with measured performance results on 2 and 3 workstations.

With the performance model extrapolations are made for a cluster of up to 16 workstations. Although the performance model is optimistic and deviates from reality as more processors are considered, the breakdown of parallel performance can be observed.

1. INTRODUCTION

In the past few years, a 3D multi-block, multi-zone Navier-Stokes solver for compressible flow has been parallelised at NLR. In this parallelisation, the sequential algorithm has been maintained. The block subdivision is being used for distributing the work over the parallel processors.

Performance results have been obtained on several parallel platforms. These performance results were achieved by porting the parallel solver to the platform and executing the flow solver for several applications. See, e.g. reference [2], where a benchmark is reported on a cluster of 3 IBM RISC System/6000 machines, model 370, interconnected through ethernet with PVM version 3.2 as message passing system.

In this modus operandi the parallel solver has to be ported to each platform and has to be executed for the applications before the parallel performance is known. Furthermore, the question '*What are the elements determining this parallel performance*' remains unanswered. For this reason, the performance is being modelled. The model gives an estimate of the performance and especially of the economical aspect of breakdown of parallel performance *before* the porting work has been done. Furthermore, it gives a theoretical reference for interpreting the performance results.

Below, a performance model is given for one specific application problem on a cluster of workstations. The application is the flow over a flat plate (see section 2). The network characteristics of the benchmarked cluster of the above described IBM workstations have been used (section 3).

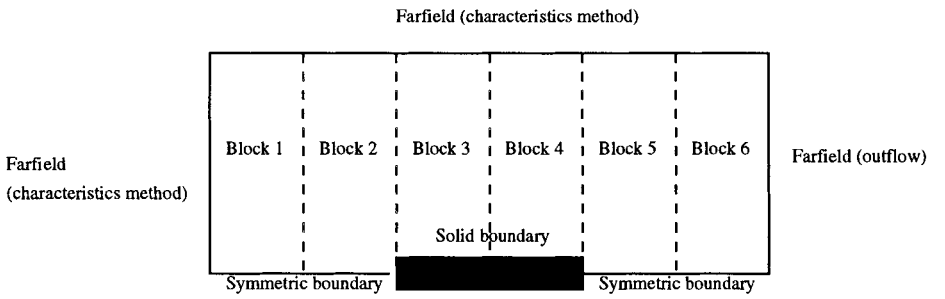


Figure 1. 6 block pipe-line; boundary conditions for flat plate problem

2. THE APPLICATION PROBLEM

The application problem is laminar flow over a flat plate. The application problem has been chosen because it represents relevant applications with easy geometry. Moreover, the application has been part of the validation of the flow solver, for comparison with the Blasius solution of the Prandtl boundary-layer equations [1]. The used grid is extremely fine for this application (384 intervals in x-direction ; 128 intervals in z-direction).

The grid is subdivided, in a pipe-line manner, into blocks with identical numbers of grid points (see figure 1). A block subdivision influences the performance in two ways: by adding calculations, and by increasing communication. See further section 4. The block subdivision is used for distributing work over the workstations.

Since the grid (G_a) is considered to be extremely fine, it is of interest to study the effect of coarser grids on the performance results. In the grid G_b the ratio of number of grid points in both directions has been preserved. In the third grid G_c, this ratio has been changed. The grid sizes are given in table 1.

3. THE PARALLEL COMPUTER

The model of the parallel computer is split up into two submodels: one model for single processor performance, and one model for the network performance. Assuming that the single processor performance is the same as in a parallel execution, the single processor performance has been determined experimentally for the specific application.

Table 1
Grid information

	grid G_a	grid G_b	grid G_c
# intervals $NI_x * NI_y * NI_z$	384*1*128	96*1*32	48*1*32
# grid points $NP_x * NP_y * NP_z$	385*2*129	97*2*33	49*2*33
# grid points	99330	6402	3234

This assumption foremost requires that the processor performance does not depend on the grid size.
The network performance model comprises a linear expression for the communication time for one message:

$$T_{comm}(x) = C_1 + C_2 * x \tag{1}$$

where x is the number of bytes to be transferred. The coefficient C_1 is the latency, and the coefficient C_2 is the reciproke of the bandwidth. The latency and bandwidth on the cluster of work stations are measured for a busy network.

4. THE ALGORITHM

The main algorithm of the block-structured solver consists of iterations over a mixture of calculations and communications (see figure 2).

4.1. Calculations

It is assumed that the calculational work scales with the number of iterations neglecting start-up and close-down of the execution:

$$W^b(It) = It * W^b \tag{2}$$

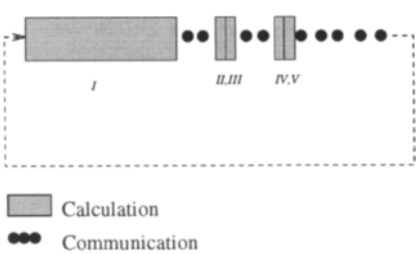
where $W^b(It)$ is the sum of the calculational work for It iterations, b is the number of blocks, and W^b is the sum of the calculational work per iteration.

In the flow solver, the block subdivision is used to distribute the work over the work-stations. Therefore, the calculational work W^b will be distributed over b processors. In the ideal case, on a parallel platform, the calculational work on p processors is equal to the calculational work on a single processor, and the work W^p is distributed evenly over the p processors. The assumption that the calculational work W^p is distributed evenly over the p processors is equivalent to the assumption that the calculational work W^p is distributed evenly over the p blocks. So, ideally,

$$W^p = W^1 \tag{3}$$

and, when all processors have the same computation speed, the calculation time T^i_{calc} is

$$T^i_{calc} = It * C^s_{speed} * W^1/p \tag{4}$$



FOR all iterations DO
Apply conservation laws and boundary conditions (I)
Average flow on block boundaries (II)
Apply boundary conditions (III)
Copy flow in overlap regions (IV)
Calculate convergence information (V)
ENDFOR

Figure 2. Calculation - Communication pattern

The numbers for task II on the grids G_b and G_c are equal because the number of grid points in normal direction are the same (see table 1). In the parallel algorithm, the averaging is performed for each internal boundary on a processor. Therefore, if two neighbouring blocks are on different processors, the averaging over their interfacing boundary is executed twice. In such a case, the actual W_{II} doubles. Still, the work W_{II} takes far less than 1 % of the work W_I, and is neglected.

When internal block boundaries are added, task III increases because it is applied twice in the grid points on the internal boundary. The work W_{III} grows with about $(p-1)/NP_x$ in the middle blocks. In the two end blocks, some of the work W_{III} scales in this same way, the remainder stays constant: the work W_{III} increases at worst with about $(p-1)/NP_x$ in the end blocks. For the 6 block pipe line, W_{III} is at the most a few percent of W_I. And the dependency on the number of processors is such that the percentage does not grow when the number of processors increases. The work in task III is neglected.

The work in task IV is small as compared to task I. It is neglected.

In summary, work W^6 on a pipeline of 6 blocks is increased with respect to work W^1 in a single block. The most significant increase comes from task I and is about 1 % on the finest grid and about 5 % on the coarsest grid.

Using W^6 as an estimate for W^1, formula (4) for the calculation time becomes:

$$T_{calc} = It * C^s_{speed} * W^6/p \tag{6}$$

By executing the 6 block pipe-line on a single processor we see that the sustained performance C^s_{speed} does not depend on the various grid sizes (table 2). From this, it is concluded that this sustained performance applies even if the grids are distributed over several processors.

4.2. Communication

The communication points in the algorithm are given in figure 2. In the above model for the computational part of the algorithm, there is one large batch of computations, and followed by neglected small batches of calculations. The better the work for the large batch of computations is balanced, the more synchronised all processors will start communicating: all messages will virtually hit the network at once.

On this specific network, the messages are handled sequentially. This gives that the time for communication is (see Eq. 1):

$$T_{comm} = \sum_{all\ messages} C_1 + C_2 * x(message) \tag{7}$$

Messages are used to communicate average information (between tasks *I-II*), the flow in the overlap regions (between tasks *III-IV*), convergence information (after task *V*), and synchronisation information.

The numbers of messages and total lengths of messages for a pipe-line of blocks on 3 processors has been counted (see table 3). For the separate processors, the communication with the neighbouring blocks depends only slightly on the boundary conditions in the blocks.

When p processors are used, there are $p-1$ internal boundaries over which average information and overlap information is communicated. Both the numbers of messages

Table 2
Measured work and performance

	unit	grid G_a	grid G_b	grid G_c
Measured				
Work/iteration W_I^6	MFLOPs/it	389.7	27.92	15.74
Work/iteration W_{II}^6	kFLOPs/it	65.27	17.25	17.25
(5 internal boundaries)				
Work/iteration W_{III}^6	kFLOPs/it	1824.2	462.98	334.42
Work/iteration W_{IV}^6	kFLOPs/it	$\ll 1$	$\ll 1$	$\ll 1$
$W_I^6 + W_{II}^6 + W_{III}^6 + W_{IV}^6$	MFLOPs/it	391.5	28.34	16.09
$20{*}W^6$ + Start-up + Close-down	MFLOPs	8104.1	586.3	331.8
Model				
Approximation W^1	MFLOPs/it	405.2	29.32	16.59
Explained		96 %	96 %	96 %
Measured				
Execution time on 1 processor				
$50W^6$	s	-	557	317
$5W^6$	s	769	-	-
Sustained performance	MFLOPs/s	2.63	2.63	2.62

where C^s_{speed} is the sustained computation speed.

Below, the assumptions underlying the model for the calculation time (Eq. 4) are verified for a pipe-line of 6 blocks with identical numbers of grid points per block. The verification figures are used to derive the sustained single processor computation speed C^s_{speed} and to check that this is independent of the grid sizes.

The calculational work per iteration, W, is the sum of the work in the five calculational tasks (figure 2):

$$W = W_I + W_{II} + W_{III} + W_{IV} + W_V \tag{5}$$

When the computational flow domain is a single block, the calculational parts *II-IV* completely vanish: $W_{II}^1 = W_{III}^1 = W_{IV}^1 = 0$.

For the pipe-line of 6 blocks, the numbers of FLOPs are counted for tasks *I-IV* (see table 2). With these tasks, over 96% of the total number of FLOPs are explained.

The numbers of FLOPs/iteration in the separate blocks differ less than 1 % from average on the finest grid and about 5 % on the coarsest grid. Therefore, this application problem agrees reasonably with the ideal of 'even distribution'.

When increasing the number of blocks in the pipe line for the distribution of the work over an increased number of blocks, internal block boundaries are added. The effect of increased number of blocks on the calculational tasks *I-IV* is discussed now.

Because the algorithm is the same cell-vertex scheme as in the sequential solver, task I is applied twice in the grid points on the internal boundary. Therefore, the work W_I increases with about $(p-1)/NP_x$. For the 6 block pipe line, this amounts to little over 1 % on the finest grid, and over 10 % on the coarsest grid.

Table 3
Numbers and total length of messages

	# occurrences per iteration	grid G_a	grid G_b	grid G_c
Measured				
I-II	$p-1$			
# messages		12	12	12
# bytes		10560	2880	2880
III-IV	$p-1$			
# messages		14	14	14
# bytes		52352	13952	13952
V-end	$p-1$			
# messages		4	4	4
# bytes		188	188	188
synchronisation	$p*(p-1)$			
# messages		1	1	1
# bytes		8	8	8
Model				
# messages	$p*(p-1)$	1	1	1
# bytes	$p*(p-1)$	0	0	0
# messages	$p-1$	30	30	30
# bytes	$p-1$	63000	16900	16900

and the lengths of the messages are incorporated in the model for the communication time.

All processors except one communicate their local convergence information to the designated I/O processor. The length of the messages is such that for the considered network characteristics, their contribution to the communication time is negligible. The number of messages is relevant, however.

Finally, a synchronisation signal is sent by each processor to each other processor. Again the lengths of the messages give negligible contribution to the communication time.

When the number of processors increases, the numbers of messages per internal interface for all of the communications will remain the same, as will the lengths of the messages.

5. THE PERFORMANCE MODEL

Because of the considerations in subsection 4.2, the calculations and communications are executed sequentially. So, the total execution time is the sum of the calculation and communication times:

$$T_{exec} = T_{calc} + T_{comm} \tag{8}$$

The calculation time T_{calc} for p processors is given by Equation 6. The coefficients C^s_{speed} and W are given on table 2.

The total number of messages per iteration and the total amount of bytes communicated are given in table 3. The net work characteristics of cluster of workstations are measured

Table 4
Estimated and measured execution times on p workstations

	unit	p	grid G_a	grid G_b	grid G_c
Estimated T_{exec}/It	s/it				
		1	154	11.1	6.31
		2	77.3	5.76	3.34
		3	52.0	4.10	2.49
Measured T_{exec}/It	s/it				
		1	154	11.1	6.34
		2	77.4	5.76	3.37
		3	53.9	4.01	2.45
Estimated T_{comm}/T_{exec}					
		1	0.	0.	0.
		2	.0040	.033	.057
		3	.012	.094	.16

for a busy network, giving a latency of .0045 s, and a bandwidth of .375 Mbyte/s ([3])

The estimated execution time per iteration as a function of the number of workstations is compared with measured execution times (see table 4). Because the available network consists of 3 workstations, the comparison is limited to 2 and 3 processors. The comparison shows reasonably good agreement. Note that on the fine grid G_a, in fact, only calculation times are compared. On the two coarser grids G_b and G_c, the communication time is a visible part of the execution time.

The complete breakdown of parallel performance is observed when the use of an additional processor gives decreased speed-up, i.e.

$$dSp/dp \leq 0 \tag{9}$$

The derivative dSp/dp as following from the performance model is given in figure 3 for the three grid sizes. On the coarser grids G_b and G_c, the number of usefull processors is 6, resp. 5. On the finest grid G_a, the number of usefull processors is 18.

6. CONCLUDING REMARKS

A simple performance model has been presented. The model has been validated for performance on 2 and 3 workstations, with satisfying agreement.

For the application, the number of messages is the dominating contribution to the communication time. An investigation whether the number of messages can be reduced, can certainly result in better parallel performance. Because the number of messages is far more important for the communcation time than the total length of the messages, the pipe-line of blocks gives the best distribution of the work over the processors.

For the fine grid G_a, the complete breakdown of parallel performance occurs at 18 workstations. For the coarser grids, the parallel performance breakdown occurs at 5-6 workstations.

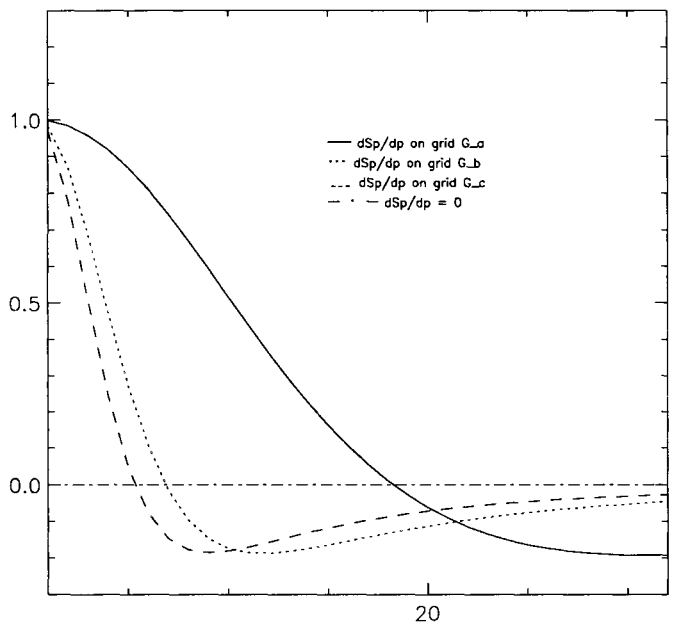


Figure 3. *dSp/dp* as a function of the number of processors

Acknowledgement

The support of J.P. Geschiere in porting the program to the cluster of workstations is gratefully acknowledged.

REFERENCES

1. D. Dijkstra, J.G.M. Kuerten, *An easy test-case for a Navier-Stokes solver*, Proceedings of the First European Computational Fluid Dynamics Conference, 7-11 September 1992, Brussels, Belgium, Volume 2, pp 977-984.
2. J.P. Geschiere, P.A. van Mourik, *Parallelising a large scale 3D multi-block Navier-Stokes solver*, NLR TP 94239.
3. A. R. Sukul, *Design of a load balancing preprocessor*, NLR TR 93463.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

Further Acceleration of an Unsteady Navier-Stokes Solver for Cascade Flows on the NWT

Takashi Yamane[a]

[a]Thermofluid Dynamics Division, National Aerospace Laboratory
7-44-1 Jindaiji-higashi, Chofu, Tokyo 182, JAPAN
E-mail:yamane@nal.go.jp

An unsteady Navier-Stokes solver for cascade flow problems, which was developed on the Numerical Wind Tunnel (NWT) system, has been modified for further faster calculation speed. The new code uses multiple processor groups for multiple calculation regions thus 70 times faster speed has been obtained by 120 PEs consisting of five 24-PE groups.

1. INTRODUCTION

Requirements of many CFD researchers can be classified into two kinds; one is to increase the number of grid points and the other is to calculate fast using the same size of computational mesh. Parallel programmings are not easy works for general researches who are not necessarily computer specialists but satisfying the first desire is not so difficult because once the program has been developed the number of grid points can be increased as much as possible until it reaches the limit by the hardware of the computer system. However, for those who just want a fast calculation speed it is very hard because the mesh size usually decides the maxmum number of processors.

The auther has developed a code for cascade flow problems of turbomachines which splits one blade passage by multiple processor elements (PE) of the Numerical Wind Tunnel at National Aerospace Laboratory[1]. The problem solved by this code was an unsteady interaction flow field of impeller and diffuser of a centrifugal compressor. Each calculation region was splitted by multiple processors in order to make a code applicable for various problems from single blade pitch flows to unsteady interactions of rotor-stator stage, thus each calculation region have been solved sequentially. By this code up to 16 times faster speed has been obtained using 24 PEs but it was nearly a limit due to the number of mesh lines to be splitted. However, 24 PEs are only 17% of total calculation power of NWT, so the calculation speed can be get much faster.

This paper reports a further acceleration of the code in solving unsteady interaction flow of impeller/diffuser in a centrifugal compressor and introduces some numerical results including comparisons with experiments.

2. PARALLELIZATION METHOD AND CALCULATION SPEED

The computer code in this study has been developed to solve unsteady flow phenomena of impeller-diffuser interaction in centrifugal compressors. The numerical method of the original three dimensional Navier-Stokes solver is based upon Chakravarthy-Osher's TVD scheme and

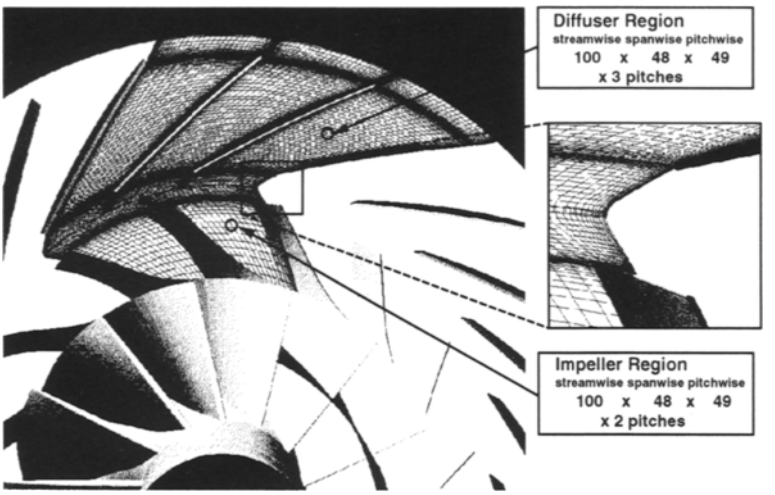


Figure 1. Computational Grid for Centrifugal Compressor

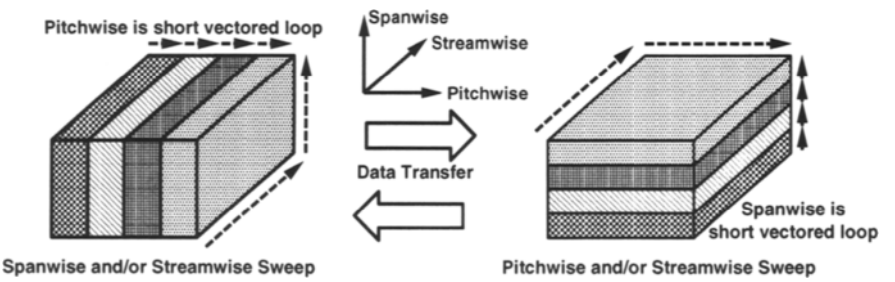


Figure 2. Decomposition Pattern inside One Region

differentiable flux limiter is employed in stead of the "minmod" function. LU-ADI approximate factorization method is applied for the time integration and Baldwin-Lomax algebraic turbulence model is also used.

As shown in Figure 1, at least 2 pitches for impeller and 3 for diffuser are required for fully unsteady simulations. An economical way to obtain stage performances is to average along the rotor/stator interface circumferentially which requires only 1 impeller and 1 diffuser pitch.

In programming on the NWT, users should be aware that it is a cluster of vector processors. As shown in Figure 1, the spanwise mesh lines are 48 which are divided by 24 PEs at maximum and the streamwise and pitchwise DO-loops can be calculated fastly by the vector processors (Figure 2). For spanwise DO-loops the region is re-splitted pitchwisely to avoid short vectored loops. Some delay may occur due to the data transfer between two types of splitting but the calculation speed in one region thus becomes fast. The splitting here is one dimensional and

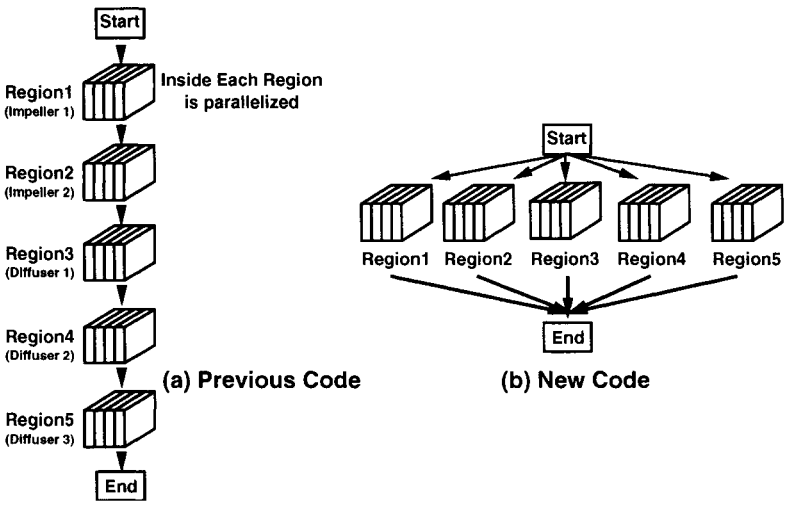


Figure 3. Concept of Multiple PE Group Calculation

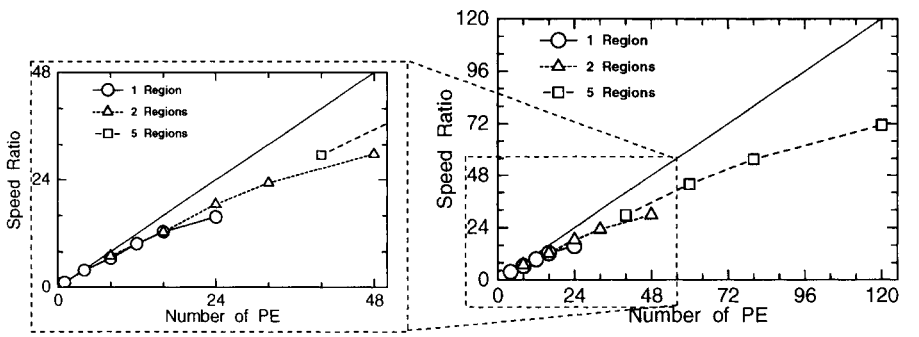


Figure 4. Speed Ratio

three dimensional decomposition of one calculation region may increase the number of usable PEs, however, it will result in many short DO-loops which become a disadvantage for vectorized calculations.

In the previous program, this calculation process in one region has been executed sequentially for 5 regions consisting from 2 impeller pitches and 3 diffuser pitches by single PE group (Figure 3(a)). The calculation speed was satisfactory but only 24 PEs out of available 140 have been used, so there were capability for faster calculation with much more PEs. The algorithm is quite simple: calculate all regions simultaneously using multiple processor groups (Figure 3(b)). However the actual program has turned out to be a quite complicated one.

In Figure 4, the speed ratios compared with the calculation time by 1 PE have been plotted against the number of PEs. Steady stage results by 2 PE groups and fully unsteady results by 5 PE groups are compared with speed done by 1 PE group. As the number of PEs in a group increases the performance becomes worse for all cases, but 2 group speed is about 1.9 times of 1 group and 5 group is 4.4 times generally, thus approximately 70 times faster speed than one PE speed has been obtained using 120 PEs.

3. RESULTS

3.1. Time Averaged Experimental and Computational Results

Figure 5 shows the performance map of the model centrifugal compressor which is obtained from experiments, unsteady calculations, and steady stage calculations. The exit total pressure has been measured at the exhaust duct and at just after diffuser vane, so the differences in pressure ratio are due to losses at the scroll and duct. At large volume flow rate conditions, the flow at the diffuser exit becomes transonic and quite unsteady, thus the curves show slight oscilations at lower ends.

Most experimental observations and numerical simulations are performed for the operating point along the performance line of 70000 rpm. Due to the high revolutional speed, instantaneous measurements are very difficult, thus the experimental results are steady or time averaged. Numerical simulations have been performed for unsteady and steady stage cases under the same exit static pressure values, whence the exit total pressures have been calculated by averaging at the diffuser exit boundary.

In terms of pressure ratios, numerical values and experiments show a good correlation except some differences between unsteady and steady calculations. However, numerical mass flow rates show larger values by approximately 10% which may be explained by the existance of 3 m

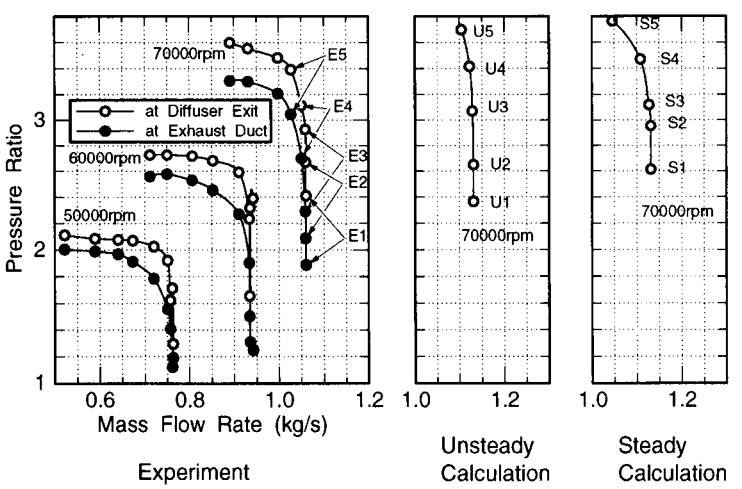


Figure 5. Performance Map of Model Centrifugal Compressor

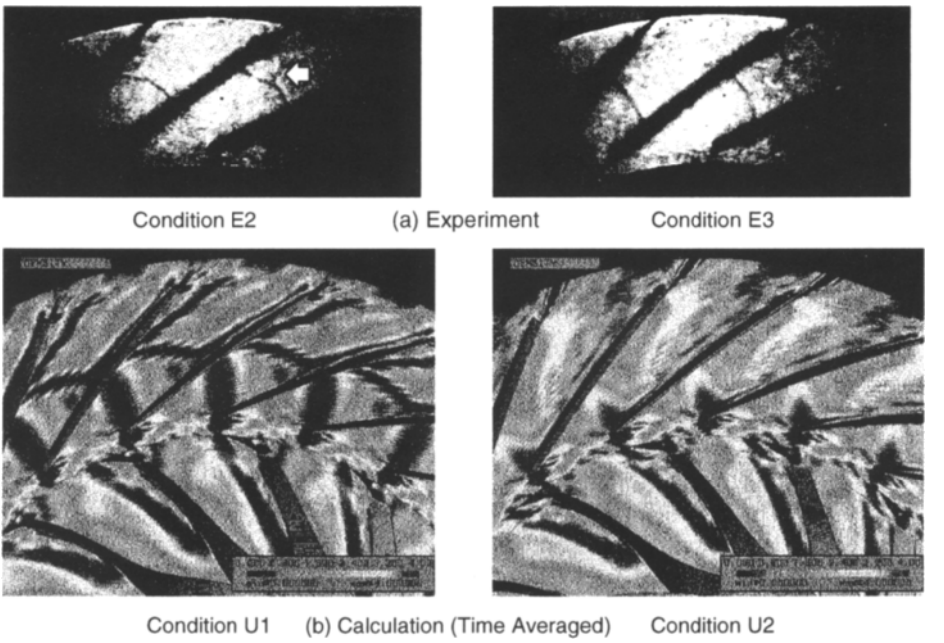


Figure 6. Schlieren Pictures

long inlet duct that has not been taken into account in simulations.

Schlieren pictures by experiments around the diffuser inlet region taken at condition E2 and E3 are shown in Figure 6(a). These are of conditions where volume flow rate is very large, so the flow at the diffuser vane throat tends to be choked and transonic. In order to compare with these pictures, the time averaged results of unsteady calculations have been processed and drawn in Figure 6(b) so that they look like schlieren photographs. (Dark shadows indicate the locations of large density gradients.) At the condition E2, an arch shape is observed, which can be also seen at the same location in the computation U1, but a branch extending toward the diffuser exit from the arch (marked with a white arrow in the figure) is not clear in the computation due to the time averaging. The numerical schlieren figure shows additional complex shadows : the one along the impeller blade which indicates the boundary of jet and wake and an expansion wave from the diffuser leading edge toward the pressure surface of neighboring vane. As the exit pressure rises, the arch shape shock wave moves towards upstream (E3) and then disappears. At the calculation condition U2, the flow is not always choked at the diffuser throat, thus the shock wave has already disappeared.

Static pressure distributions on the shroud wall of the diffuser inlet region were measured along the performance line of 70000 rpm (Figure 7(a)). At operating condition E2 where volume flow rate is the largest, distorted contours due to choke can be recognized around the throat of diffuser vanes. As the exit pressure goes higher, volume flow rate becomes smaller and contours show a smooth increase towards the diffuser exit (condition E4, E5). These pressure contours

can be compared with time averaged numerical results of Figure 7(b). At condition U1 the arch shape shock mentioned before is clearly observed where in figure of E2 is not. It is obvious from the schlieren picture (Figure 6) that there is a shock wave, so that the reason of this discrepancy must be a lack of the numbers of pressure taps in the experiment. Figures at condition U2, U3, and U4 show similar pressure pattern changes with experimental condition E3, E4, and E5.

3.2. Unsteady Numerical Results

Fully unsteady impeller/diffuser-vane interaction calculations have been executed at five operating conditions shown in Figure 5. The time step has been fixed so that the impeller revolves for its one pitch angle after 6000 step calculations. 10 out of 48 spanwise grid points on the impeller blade tip, which correspond to 2 % of blade height, have been treated such that flow can go through the blade thickness in order to represent tip clearance effect.

Figures 8 show instantaneous density contours on the midspan plane at operating condition U1. Each picture is drawn by every 1000 time step interval during which the impeller rotates for 1/6 of blade pitch angle. The flow is choked at the diffuser vane throat because of a large volume flow rate and a shock structure appears downstream of the throat. An existance of the branch which has been observed in the experimental schlieren picture (Figure 6) slightly appears in some time phases.

The wake region which looks nearly steady expands along each impeller suction surface, and dense contour lines extending from the impeller trailing edge can be seen which is the boundary of jet and wake. Inside the wake, the velocity relative to the impeller is nealy equal zero but it exceeds the speed of sound when observed from the stationally axis. This boundary line flows into diffuser vanes like a traveling wave and the incidence at the vane leading edge is fluctuating due to this wave but it is dissipated quickly before it reaches diffuser vane throat.

4. CONCLUSION

A parallelized unsteady Navier-Stokes solver for cascade flow problems has been modified for faster calculation speed. The new code yielded 70 times faster speed by using 120 PEs for unsteady simulations of a centrifugal compressor stage with 5 calculation regions. Numerical results revealed unsteady flow phenomena at various operating conditions and comparisons with experiments showed very good correlations.

REFERENCES

1. Yamane,T., "The Transplantation of an Unsteady Navier-Stokes Solver for Cascade Flows onto the NWT System", Proceedings of Parallel CFD '94

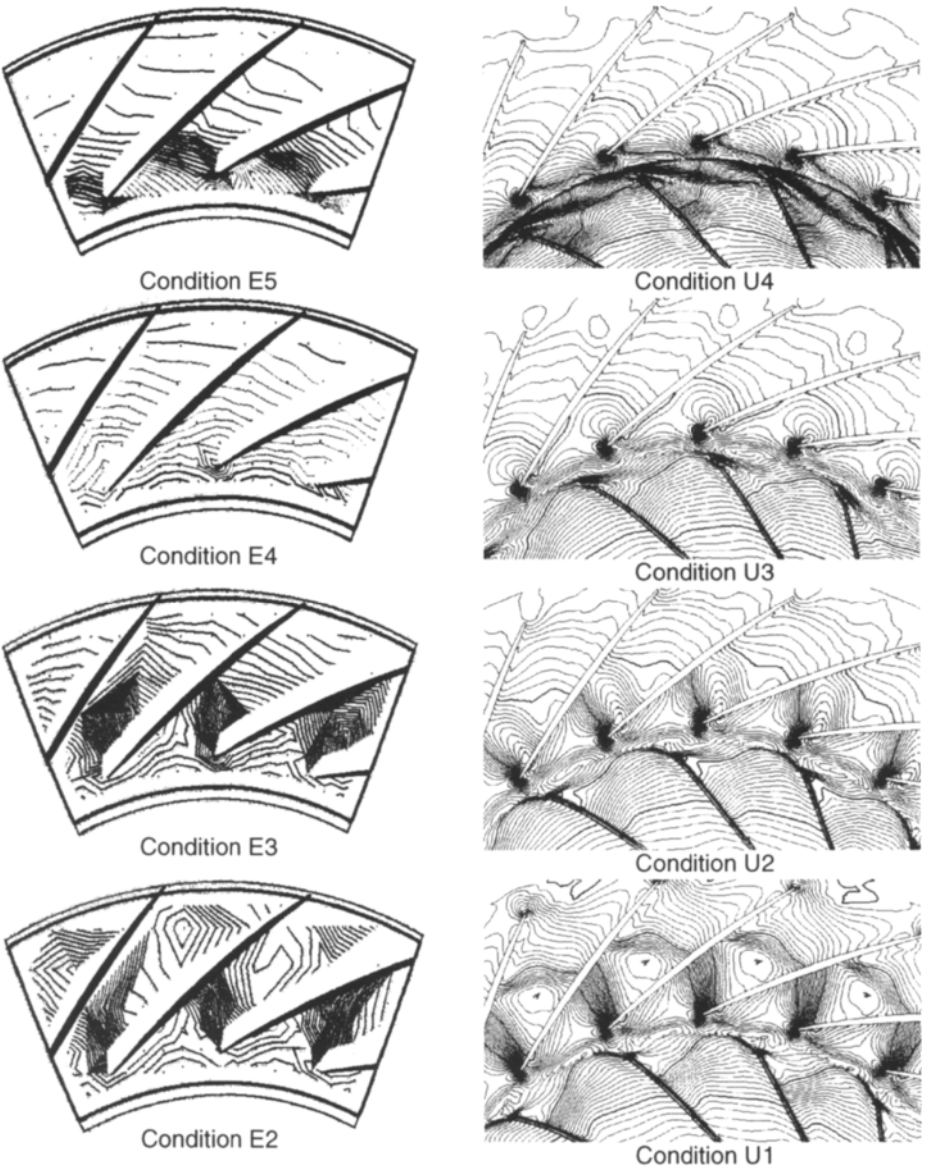


Figure 7. Static Pressure Contours on Shroud Wall

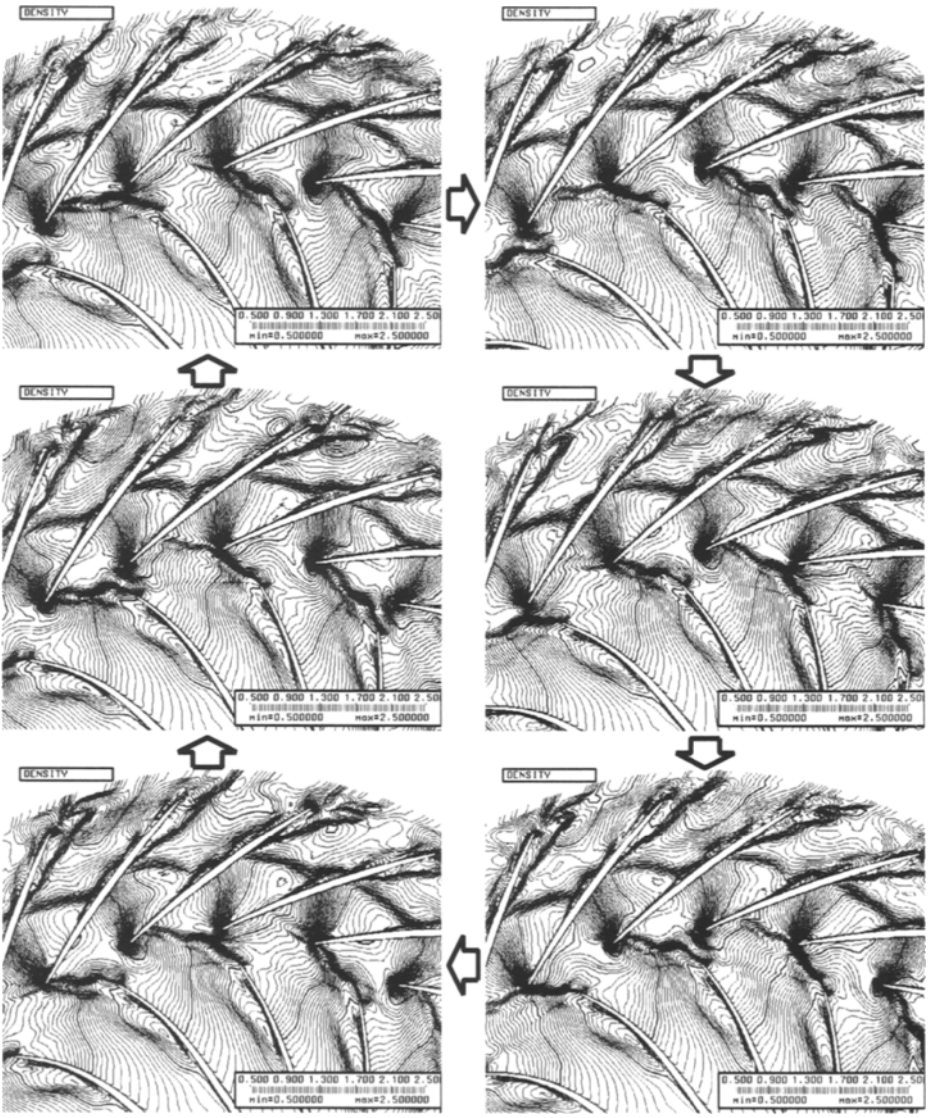


Figure 8. Instantaneous Density Contours on Midspan Plane (Condition U1)

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Load Balancing Strategy for Parallel Vortex Methods with Distributed Adaptive Data Structure

Y. L. Shieh and J. K. Lee[a], J. H. Tsai and C. A. Lin[b] *

[a] Department of Computer Science, National Tsing Hua University, Hsinchu, TAIWAN 30043

[b] Department of Power Mechanical Engineering, National Tsing Hua University, Hsinchu, TAIWAN 30043

This paper addresses the implementation of vortex filament methods with adaptive data structures on parallel machines with distributed memory to simulate three-dimensionally evolving jet. The implementation is conducted to experiment with idea with parallelism, data distribution of filament-segments, the dynamical growth of segments in the runtime and load-balancing schemes.

1. INTRODUCTION

Jet flow which contains abundant flow motions, Kelvin-Helmholtz instability, roll-up, pairing, and breakdown[1], can be found in various industrial applications. Concomitant with the development of jet propulsion in past decades, these phenomena, which affect mixing and also generate noises, have received considerable investigations. The evolution of three-dimensionally periodical jet undergone different perturbations is often employed to study these effects. Due to the nature of the vorticity induced flow problem, the vortex filament method[2][3] which approximates the vorticity field by numerous vortex filaments with an assumed vorticity distribution around the filament centreline, is adopted. When the vortex filament is severely stretched, due to the Kelvin-Helmholtz instability excitation, to the extent that the segment is unable to resolve the curvature of the filament, the segment is bisected into two segments. Therefore, the number of segments in a filament is adaptive and is dynamically growing according to the requirement of numerical accuracy in order to simulate the physical environment.

This adaptive data structure poses load imbalance in simulating the flow on Parallel machines if static data-partition method is employed. Present parallelization of the vortex filament method is designed on the native SPMD C environment and this paper presents an experimental report with various techniques in filament-segment data distribution using packet-oriented data structure within computations to achieve good performance

*The work documented herein was supported by National Science Council of Taiwan under grant NSC-83-0401-E-007-011 which the authors gratefully acknowledge. Gratitude is also expressed to the National Centre for High Performance Computing, Taiwan, for providing access to its 8-node Dec Alpha workstation cluster and 32-node IBM-SP2 machines.

for this problem.

2. MATHEMATICAL MODEL

The flow motion for incompressible inviscid fluid can be described as :

$$\frac{\partial \Omega}{\partial t} + (V \cdot \nabla)\Omega = (\Omega \cdot \nabla)V \tag{1}$$

where Ω and V represent vorticity and velocity, respectively.

Vortex filament method approximates the vorticity field by numerous vortex filaments with an assumed vorticity distribution around the filament centrelines. The velocity, based on the Biot-Savart law, is calculated by filaments of finite core radius, as suggested by Leonard[2], as

$$V \approx \sum_{i=1}^{N} \sum_{j=1}^{M_i} \Gamma_i K_\sigma (X - X_{i,j}) \delta l_{i,j} \tag{2}$$

$$K_\sigma(X) = -\frac{1}{4\pi |X|^3} f(\frac{X}{\sigma}) \begin{pmatrix} 0 & x_3 & -x_2 \\ -x_3 & 0 & x_1 \\ x_2 & -x_1 & 0 \end{pmatrix}, \qquad f(r) = \frac{r^3}{r^2 + \alpha^{3/2}}$$

where α is a numerical parameter depending on the vorticity distribution of vortex core. In this paper, the vorticity distribution is assumed to be constant and Gaussian and α is selected to be 0.413[4].

Spatial curve integral is performed by cutting every filament into many line segments short enough to express the curvatures of these filaments. The length of every segments is represented by $\delta l_{i,j} = |X_{i,j+1} - X_{i,j}|$, where $X_{i,j}$ indicates the adjoint point between segments. Trajectories of each adjoint points can be obtained by integrating,

$$\frac{dX_{i,j}}{dt} = V(X_{i,j}, t) \tag{3}$$

with time. In the present methodology, second-order time scheme is used to move the adjoint points. When the vortex filament is severely stretched to the extent that the segment is unable to resolve the curvature of the filament, the segment is bisected into two segments. Here, we followed the algorithm by Kino & Ghoniem[5], where the segment is bisected equally when this segment is longer than the longest segment at initial time.

3. PARALLEL ALGORITHM

The data structures are composed by a group of rings distributed among three dimensional spaces, with each ring consisting of a collection of filament segments. The number of segments in a ring is adaptive and is dynamically growing according to the the requirement of numerical accuracy in order to simulate the physical environment. Figure 1 shows an instance of the data structure of our problem requires at a given time. It represents an irregular aggregate structure with non-rectangular index sets. The number of rows of the structure is corresponding to the number of rings in the physical domain. The length of each row is related to the number of segments in a ring.

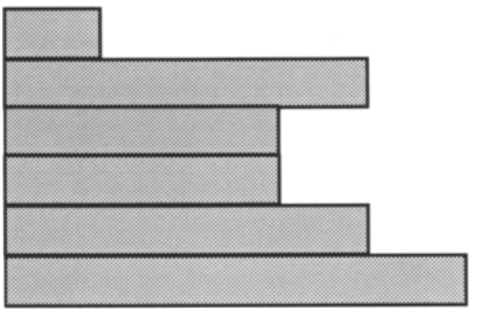


Figure 1. Irregular aggregate structures with non-rectangular index sets.

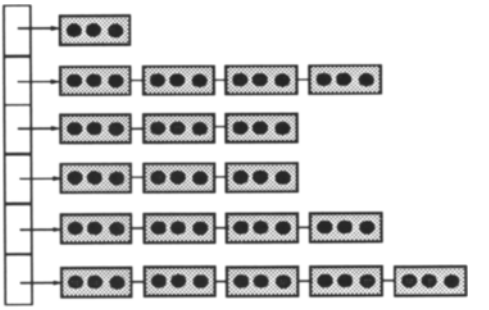


Figure 2. A Packet-Oriented Load-Balancing Data Structures.

The traditional approach[6] uses simple "BLOCK" style partitioning scheme. In that approach, the whole data structure is partitioned into an set of abstract structures called "fragment". A fragment is made up by a set of subrings. Each subring is a set of contiguous segments in a ring. The group of rings are partitioned into fragments and distributed among processors. There are two problems with "BLOCK" style partitioning scheme in HPF to support the structure in our problem. First, the number of segments in our structure increase dynamically, and data are needed to be inserted into the array. The block-styled array implementation requires all the data following the insertion point to be moved back. Second, and the most importantly, the irregularity of the particle growth results in load un-balancing so that the performance of parallel programs deteriorate. To solve the above problems, we propose a novel technique called "packet-oriented" data structure which can work well with dynamically growing data structures.

3.1. Adaptive Data Structure with Load-Balancing Schemes

A packet-oriented parallel data structure, shown in Figure 2, is proposed by us to support dynamically growing structures with non-rectangular index sets. In the scheme, the array is divided into a set of packets which are then distributed among processors and the balance of the computation is controlled by the number of the packets in every process. When data are inserted into an array, they are actually inserted into the packet. The number of elements of a packet grows with the data and, in our construction, the packet is automatically split if the number of elements in the packet is over a preset bound. Packets are moved around different processors to keep the balances of computational loads among processors. Figure 3 shows that in the global view each processor possesses a portion of rings. The structure is very much like a distributed array[7] except that it allows the size of the distributed to be grown and works comparable with a set of load-balancing algorithms.

Arrays are distributed among processors by dividing into packets. When the imbalance begins, the work load is rebalanced by dividing packets equally among processors at runtime. To support this kind of abstract data structure, a set of operations for packets is

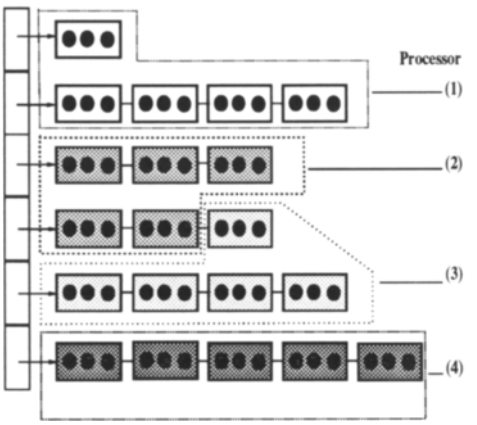


Figure 3. Global View of Packet-Oriented Partitioning Scheme.

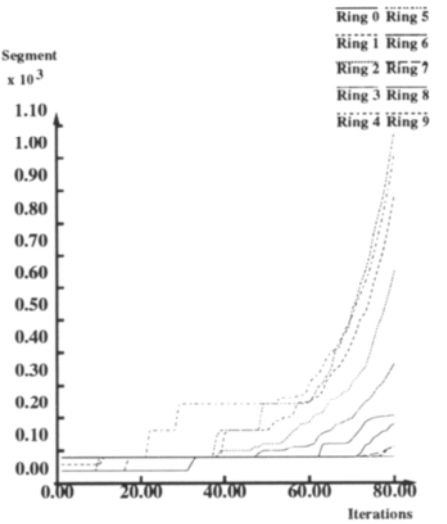


Figure 4. Segment number distribution on different ring

constructed and is listed in Table 1. The AddPack and DelPack control the caching of the remote data. SendPack and RecvPack transfer the packets between different processors. IsPackStart and IsPackEnd are used to control the beginning and ending of a packet.

4. RESULTS AND DISCUSSIONS

A circular, inviscid, spatially periodical jet flow subjected to axial perturbations forms the basis of the computations. The primary vortical structure of the jet is observed[3] to be dominated by inviscidity and spatial periodicity further simplifies the calculations of the jet development. Radius of the jet is taken to be R=5 and a constant filament radius of σ =0.1R is adopted. Initial axial perturbation is simulated by a sinusoidal variations

Table 1
Pack Relative Function

GetPack(ArrayX,Index)	Return the address of the packet contained the element of the index.
IsPackEnd(ArrayX,Index)	Given an index and decide if it is the end element of a packet.
IsPackStart(ArrayX,Index)	Given an index and decide if it is the starting element of a packet.
SendPack(ArrayX,Index)	Given an index and send the packet to the processor who needs it.
RecvPack(ArrayX,Index)	Given an index and receive the packet from the processor.
AddPack(ArrayX,PackY)	Given a packet and add it to the ring.
DelPack(ArrayX,PackY)	Remove a packet from a ring.

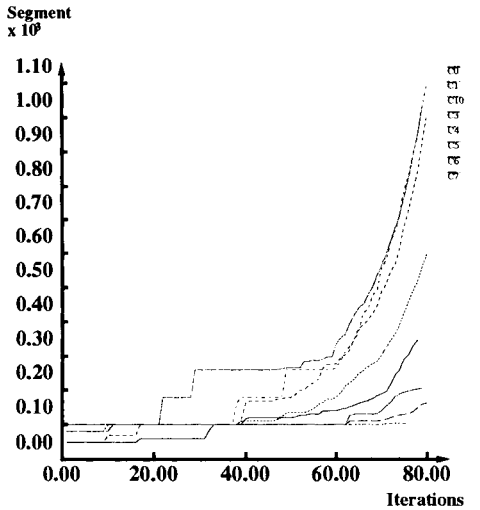


Figure 5. Number of segments in 8-node machine with static partition scheme

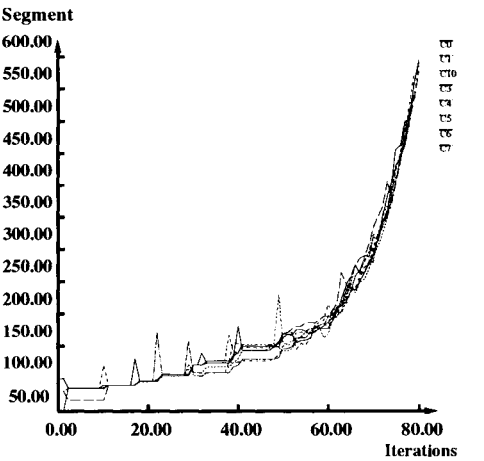


Figure 6. Number of segments in 8-node machine with run-time load-balancing scheme

of circulation, the undisturbed circulation is Γ_0=5.0,

$$\Gamma = \Gamma_0(1 + \epsilon SIN(2\pi z/\lambda))$$

where λ is the axial wavelength, chosen as 2π and ϵ=0.05, is the perturbation strength.

Mathematically, the computation should be applied on infinite periods. However, preliminary computations indicated that negligible variations of the velocity profiles were observed provided at least three upstream and downstream images are included. Therefore, we approximated the infinite periods with three upstream and downstream periods, and discretized the axial wavelength into 10 filaments, each of which initially contained 40 segments.

Due to the Kelvin-Helmholtz instability excitation, the vortex filament is severely stretched and in order to preserve accuracy, the lengthened segment is consecutively bisected into two segments. This phenomenon can be clearly seen from Figure 4 which shows the segments number on each ring indicating the irregular segment growth pattern.

Since the computational loading is proportional to the distributed segments number on the individual processor, it is essential to maintain the number of segments on separate processor equal. While the runs with run-time load balancing, shown in Figures 6 and 8, demonstrate the superiority of the method to achieve load balance, Figures 5 and 7 indicate the extent of loading unbalance on different processor for the eight and sixteen processor runs with static data partition.

Attention is now directed to the CPU time performance on different machines. Table 2 shows the normalised accumulated cpu time on the Dec Alpha workstation clusters, IBM

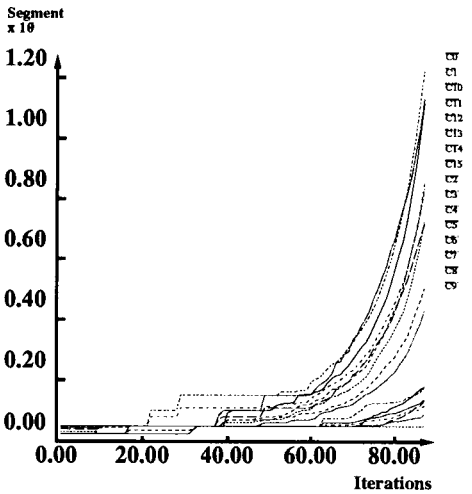


Figure 7. Number of segments in 16-node machine with static partition scheme

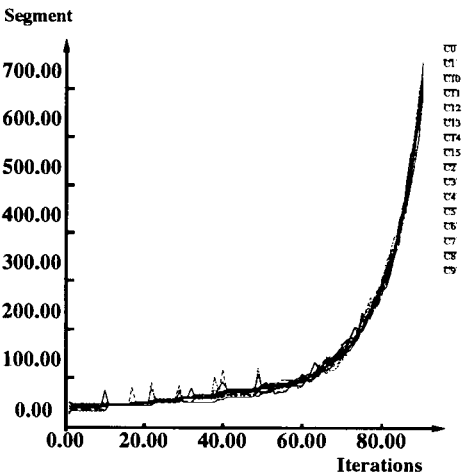


Figure 8. Number of segments in 16-node machine with run-time load-balancing scheme

SP2 and nCUBE2 upto fifty iterations. The superiority of the load balancing can be further affirmed by the results, as shown in table 3 which indicates the ratio of the execution time of the unbalance and balance runs. It should be pointed out that the Dec Alpha workstation clusteres and IBM SP2 are not dedicated machines, therefore the advantage of run-time load balancing on these machines, though favourable, is obscured by the nondedicated environment.

Maximum speedup of 2.6 was achieved using the nCUBE2 machine which is a dedicated environment. The difference of the speed up can be attributed to how evenly the filament's segments were distributed among the processors. This might be due to the fact that the newly generated segments, due to the stretching of the vortex filament, can not be divided evenly among the processors in the form of packets which had finite number of segments in it.

In order to examine the scalability of the present scheme, focus is directed to the CPU time history of the nCUBE2 machine runs, a dedicated machine, shown in Figures 9 and 10. The advantages of load-balancing can be demonstrated on the cpu time history which again indicates the superior performance of load-balancing scheme over static block-cyclic scheme. As was indicated by the evenly distributed segments, Figures 6 and 8, good scalability of the scheme is achieved.

Table 2
The normalised execution time with and without load-balancing scheme at 50 iteration
T / T_a Iterations = 50

	DEC 3000/500		SP2 9076		nCUBE2	
	Unbalance	Balance	UnBalance	Balance	UnBanace	Balance
4 Processors	0.131	0.085	0.172	0.095	1.220	0.734
8 Processors	0.138	0.089	0.121	0.093	1	0.377
16 Processors			0.094	0.039	0.484	0.210

T_a is time spent of the program on 8 processors without balance

Table 3
The execution time ratio with and without load-balancing scheme at 50 iteration
Tu / Tb Iterations = 50

	DEC 3000/500	SP2 9076	nCUBE2
	Tu / Tb	Tu / Tb	Tu / Tb
4 Processors	1.534	1.792	1.664
8 Processors	1.547	1.311	2.647
16 Processors		2.379	2.307

5. Conclusions

The implementation of vortex methods with adaptive data structure on parallel machines is accomplished through a packet-oriented data structure to support dynamically growing data structure, the increase of filament's segments due to vortex stretching, with non-rectangular index-sets. The number of elements of a packet grows with the data and, in our construction, the packet is automatically split if the number of elements in the packet is over a preset bound. Packets are moved around different processors to keep the balances of computational loads among processors. Experiments were performed on nCUBE2, DEC Alpha work-station clusters and IBM SP2 machine. Computational results indicate that the load-balancing scheme performs much better than the static data partition scheme. It is concluded that the run-time load-balancing scheme is essential in the computations where the computational load is changing dynamically.

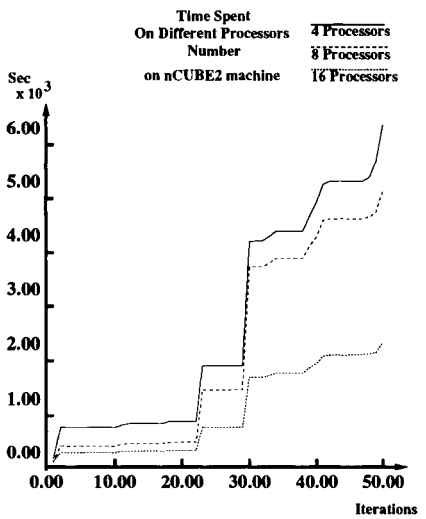


Figure 9. CPU time on nCUBE2-static data partition

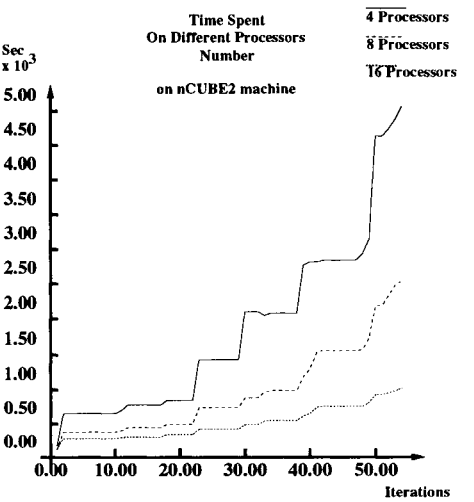


Figure 10. CPU time on nCUBE2-run-time load balancing

REFERENCES

1. Batchelor, G.K. & Gill, A.E. 1962 *Analysis of the Stability of Axisymmetric Jets* J. Fluid Mech. 14, 529.
2. Leonard, A. 1985 *Computing Three-Dimensional Incompressible Flows with Vortex Elements* Ann. Rev. Fluid Mech. 17, 523.
3. Martin, J.E. & Meiburg, E. 1991 *Numerical Investigation of Three-Dimensionally Evolving Jets Subject to Axisymmetric and Azimuthal Perturbations* J. Fluid Mech. 230, 271.
4. Ashurst, W.T, & Meiburg, E. 1988 *Three-dimensional Shear Layers via Vortex Dynamics* J. Fluid Mech. 189, 87.
5. Kino, O.M & Ghoniem, A.F. 1990 *Numerical Study of a Three-dimensional Vortex Method* J. Comp. Phys. 86, 75.
6. Y. L. Shieh, J. K. Lee, J. H. Tsai and C. A. Lin, 1994, *Computations of Three-Dimensionally Evolving Jets with Vortex Methods on Parallel Machine with Distributed Memory* Parallel CFD'94, May, Kyoto, Japan.
7. Jenq Kuen Lee and Dennis Gannon. *Object-Oriented Parallel Programming: Experiments and Results, Proceedings of Supercomputing '91*, New Mexico, November, 1991.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Portable Parallelization of the Navier-Stokes Code NSFLEX

Roland K. Höld[a] and Hubert Ritzdorf[b]

[a]Daimler–Benz Aerospace AG, Dep. Aerodynamics, LME12,
D–81663 München, Germany

[b]GMD – German National Research Center for Information Technology,
Institute for Scientific Computing and Algorithm (SCALNUM),
D–53734 St. Augustin, Germany

The paper introduces a portable parallelization of the Navier–Stokes solver NSFLEX. The solver NSFLEX has been developed by Daimler–Benz Aerospace Munich (Dasa–LM) and has been validated for many generic bodies and complex geometries in the range from subsonic to hypersonic flow field conditions. The portable parallelization is part of the German program POPINDA (Portable Parallelization of Industrial Aerodynamic Applications).

Goal of the program POPINDA is the definition and development of a communications library (CLIC) for 3–D finite–volume blockstructured CFD solvers and the parallelization of industrial applied Navier–Stokes codes. The concept of the communications library aims for a high level of portability to make parallel platforms ranging from workstation–clusters up to massively parallelized systems available for production codes. Two mastercodes (NSFLEX and FLOWer) as typical representants of production codes based on cell–centered and vertex–oriented schemes have been parallelized using the communications library CLIC in the project. The parallel codes being developed are based on highly efficient numerical algorithms (multigrid), which will allow more accurate simulations and meet increased economic, ecological and technical requirements.

This paper discusses the parallelization of the cell–centered Navier–Stokes code NSFLEX. The concept of the communications library and its major tasks will be described. Algorithmic investigations will show the influence of massively domain decomposited grids on the convergence behavior of the solver. Results for speedup on different parallel systems are presented showing superlinear scaling on cached platforms.

1. The POPINDA project

The purpose of the german POPINDA project, funded by the German Federal Ministry for Education, Science and Technology (BMBF), is the portable parallelization of aerodynamic production codes based on regular and blockstructured grids. Vertex–oriented as well as cell–centered schemes are supported to provide utilization of highly parallel systems and workstation clusters. POPINDA is a joint undertaking by Dasa Airbus (Bremen), Dasa–LM (München), DLR (Braunschweig), IBM (Heidelberg), GMD (St. Augustin) and

ORCOM (Freiberg).

The concept of parallelization is depicted in fig.1. Based on the existing Navier–Stokes codes (CEVCATS, IKARUS, MELINA and NSFLEX), two mastercodes named FLOWer and NSFLEX have been developed and parallelized within POPINDA. FLOWer is based on vertex-oriented schemes while NSFLEX uses cell-centered formulation. The parallelization of FLOWer is reviewed in [1] whereas the parallelization of NSFLEX is the subject of this paper. All of the communication tasks required to manage the parallel execution of the mastercodes and to exchange data information between processes are handled by the communications library CLIC. Therefore, the definition and development of the CLIC is a milestone within POPINDA.

2. The Navier–Stokes solver NSFLEX

The Navier–Stokes method parallelized is the finite–volume solver NSFLEX developed at Dasa–LM. It solves both the full and the thin–layer Navier–Stokes equations as well as the Euler equations [2–4].

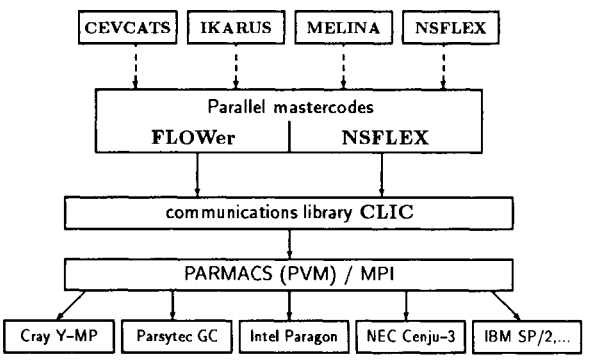


Figure 1. Concept of parallelization in project POPINDA

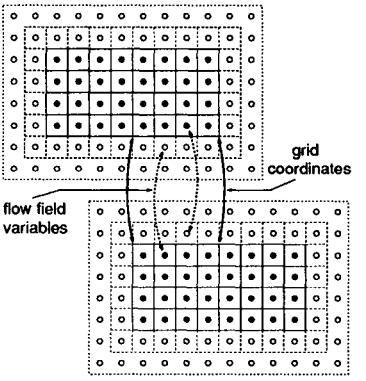


Figure 2. Multiblock technique of NSFLEX

The multiblock technique (fig. 2) used in the code NSFLEX is as follows: the grid is defined by the physical grid (solid grid lines) plus one dummy cell row (dashed lines). Within the physical grid the flow field variables are stored in the center of the volumes (solid dots). Two dummy cell rows containing flow field data (circles) are stored to handle block overlapping and physical boundary conditions. The exchange of grid coordinates and flow field data is provided by the communications library for all overlap regions of the blocks.

The governing equations of the solver are the Reynolds–averaged compressible Navier–Stokes equations in conservative form. To reach the steady state solution the equations are solved in time–dependent form. For the time integration an implicit relaxation procedure for the unfactored equations is employed which allows for large time steps [5]. A Newton

method is used, constructed by linearizing the fluxes about the known time level. The system of equations resulting from the discretization is solved approximately at every time step with a Jacobi or Gauss–Seidel relaxation. The time step is calculated with the maximum of the eigenvalues of the inviscid Jacobians. The CFL number used for calculations is typically in the order of 100. Three relaxation steps are performed at every time step to solve the linearized problem.

To evaluate the inviscid fluxes a linear, locally one–dimensional Riemann problem is solved at each volume face, whereby spatial discretization up to third–order accurate is used. A hybrid local characteristic and Steger–Warming type scheme is employed, which allows the code to work for a wide range of Mach numbers [6]. Van Albada type sensors are used to detect discontinuities in the solution. To calculate the local characteristic fluxes, the conservative variables on both sides of each volume face are extrapolated up to third order in space (MUSCL type extrapolation). This scheme guarantees the homogeneous property of the Euler fluxes, hence simplifying the evaluation of the true Jacobians of the fluxes for use on the left–hand side [2].

Because this local characteristic flux is not diffusive enough to guarantee stability for hypersonic flow cases, especially in the transient phase where shocks are moving, a hyper–diffusive flux formulation is used locally in regions with high gradients. This is a modified Steger–Warming type (flux vector splitting) flux which provides good conservation of the total enthalpy. Diffusive fluxes at the cell faces are calculated with central differences [5].

For acceleration of the solver an indirect multigrid method based on a Full Approximation Storage (FAS) concept as described in [7] is implemented. Different types of multigrid cycles are supported including a Full Multigrid Technique (FMG).

A simple approach to account for equilibrium real gas effects is incorporated which allows the Riemann solver and the left–hand side of the flow solver to remain unchanged. For more information about the equilibrium real gas modelling see [3,4,6,8,9].

At the farfield boundaries non–reflecting boundary conditions are inherent in the code since the code extracts only that information from the boundary which is allowed by the characteristic theory. At outflow boundaries, as long as the flow is supersonic, the code does not need information from downstream. In the part of the viscous regime where the flow is subsonic, the solution vector is extrapolated constantly. No upstream effect of this extrapolation has been observed to date.

At solid bodies the no–slip condition holds. Several temperature and heat flux boundary conditions are incorporated: adiabatic wall, given wall heat flux, fixed wall temperature and different levels of radiation modelling at solid surfaces [10].

3. The Communications Library CLIC

A portability interface can be defined at various levels of abstraction. One possibility is to identify high–level communication patterns in a class of applications, and to design subroutines for a central library which handle those communication tasks. An example would be a complete data exchange at subdomain boundaries in a PDE solver, rather than a single send/receive operation between two processes. If the library handles all communication requirements of the application class, the user programs themselves need not contain any explicit message passing, and are thus independent of any vendor–specific

interface. Only the library would need to be implemented for the different platforms.

At GMD this approach has been followed with the creation of the GMD Communications Library CLIC ("Communications Library for Industrial Codes", former versions are known as the GMD Comlib). The target applications are PDE solvers on regular and blockstructured grids, as they result from finite difference or finite volume discretizations. In particular, the library supports parallel multigrid applications. For this class of applications it turned out that, while the numerics differ widely, the communication sections are quite similar in many programs, depending only on the underlying problem geometry. As a consequence of the high level abstraction, the CLIC library is useful only for the application class for which it was designed.

The use of the CLIC library does not only make the applications portable among all machines on which the library is implemented. The use drastically reduces the programming effort, increases reliability and accelerates the parallelization of existing codes.

The development of the CLIC library started at GMD in 1986 with the definition and implementation of routines for 2–D and 3–D logically rectangular grids. It followed the implementation of routines for 2–D blockstructured grids. The routines for 3–D blockstructured grids are currently being developed in the project POPINDA. The routines support vertex–oriented as well as cell–centered discretization schemes.

The aim in the development of CLIC is to make programming for complex geometries as easy as for a single cube and to provide high level library routines for all communication tasks. The CLIC user interface provides the application program with all required information about the problem geometry.

The CLIC library is based on PARMACS 6.0 and, thus, is designed for a host–node (master–slave) model as shown in fig. 3. Because PARMACS is available for many parallel platforms the parallelized mastercodes can run on most platforms without vendor-specific changes. A MPI based version of the CLIC is in progress.

A host process starts the distributed application, performs input and output, and data transfers with the node processes. The host process doesn't participate in the grid computations. This is performed by the node processes. Consequently, the user application is separated in a host program and a node program. In the host program of a 3–D blockstructured application, the same input parameters are read as in the sequential versions of the mastercodes. Then, CLIC routines read in the description of the blockstructured grid, create the node processes, distribute the blocks in a load–balanced way to the allocated node processors, and finally, distribute the input parameters to the node processes. Another routine reads the grid coordinates and sends them to the corresponding node processes. After the data is distributed to the node processes, the host program calls a CLIC routine which waits for output generated by the node processes and writes that output to the corresponding output units.

Each node process executes the node program which is very similar to the original sequential program without reading the input data. The input data is transferred by CLIC routines, which receive data containing the essential block information of blocks, together with global information passed by the host program. The grid coordinates are also received by a library routine. It should be noted that a node process receives information and grid coordinates only for those blocks for which the node process performs grid computations.

Library routines also analyze the blockstructure, i.e. for each segment edge and segment

point, the adjoining blocks and the number of coinciding grid cells are determined and the edge or point is topologically classified. If the segment edge or point is part of physical boundary, the physical boundary conditions of all adjoining blocks are also determined. In addition, the grid coordinates can be examined and geometrical singularities such as block faces, which collapse to a single point, can be detected.

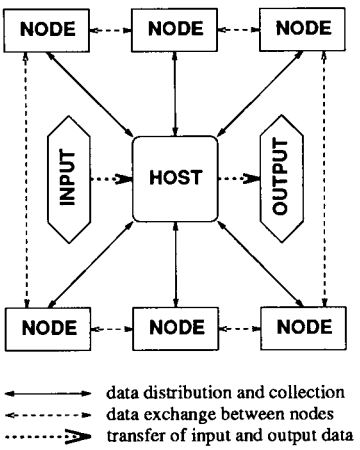


Figure 3. Host-node concept

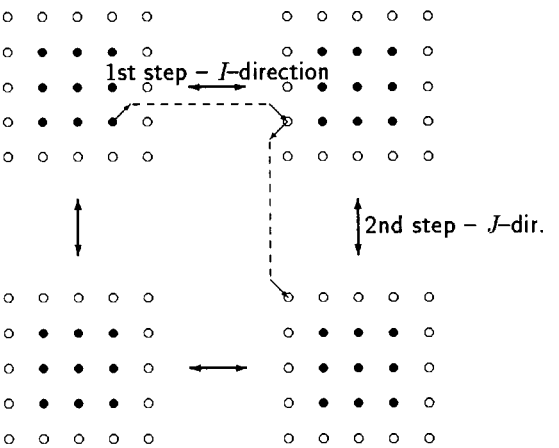


Figure 4. Updating the overlap region in a 2-D regular case

Though this data may be important for the mastercode, it is essential to the CLIC library to correctly update the overlap regions (to exchange the boundary data) of neighbouring blocks and to optimize this update procedure. An optimization of this update procedure is significant for parallel efficiency, since the corresponding CLIC routine is called most often and is the most crucial routine especially for coarse grids of multigrid algorithms. An example, for such an optimization of the update procedure is the update of regular corners of 8 blocks. The simplest technique for updating all overlap regions is to send and receive messages over all faces, edges and corners of a block; thus, 26 (6 faces, 12 edges, 8 corners) messages are sent and received per block. However, in such regular cases, the updating of all overlap regions (including the corners of the overlap regions) can be handled by just 6 messages using the following technique (fig. 4): in the first step, all blocks exchange their data with neighbour blocks in I-direction (1 message per block face); in the second and third steps, all blocks exchange their data with neighbour blocks in J- and K-direction, respectively, but now including the already updated overlap regions.

The results of the analysis of blockstructure is used to optimize the update of the overlap regions. Since it is too expensive to optimize this update sequence and to determine the areas which have to be sent to neighbouring blocks within in each update, these tasks are performed only once by CLIC routines in an initialization routine of the mastercode.

During the solution process of the mastercode, the update of the overlap regions of all blocks is then performed by calling a single CLIC routine. In that call, the user specifies the number of the multigrid levels and can choose the number of grid functions to be simultaneously exchanged.

Among other tasks, the CLIC library also performs the computation of global values (for example, global residuals), the output to files, and standard output which is generated by the node processes. In the next year, the library will be extended to adaptive blockstructures (i.e. hierarchies of blockstructures). This will include routines which create and manage adaptively refined new grid levels, which perform a load-balanced dynamic mapping and which perform all data redistribution required during adaptive multigrid algorithms.

An important fact for the development and management of the mastercodes is that there is also a sequential version of the 3-D blockstructured CLIC library. Thus, the mastercode can be sequentially executed with the same interfaces as in the parallel case.

4. Testcase definition

As a testcase for algorithmic investigations and benchmarking of the parallel code a fine grid and a coarse grid for a NACA0012 wing has been defined (fig. 5). The fine grid consists of 256 x 16 x 64 (262 144 volumes) in i x j x k direction, with j as the spanwise direction. The coarse grid is given by 128 x 8 x 32 (32 768 volumes). The algorithmic investigations presented and the benchmark tests are EULER calculations for $Mach = .63$ and $\alpha = 2.°$.

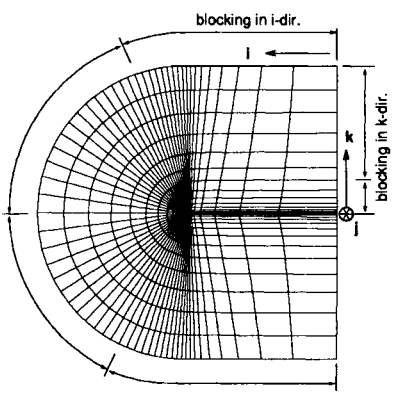


Figure 5. NACA0012 wing testcase

5. Algorithmic investigations

Different algorithmic investigations have been performed in the project POPINDA to provide information about the characteristic behavior of the mastercodes with respect to parallel applications.

To investigate the influence of massive domain decomposition on the convergence behavior of the solver the fine grid of fig. 5 has been split in i- and k-direction into 2x2=4, 4x4=16, 8x8=64 and 16x16=256 blocks. The results appear in figs. 6 and 7.

All calculations are based on a point Gauss-Seidel relaxation. Full multigrid has been applied using 3 grid levels up to 64 blocks and 2 grid levels for the 256 blocks testcase. To avoid divergence during iteration the relaxation parameter (r=.5) had to be reduced for the 256 blocks testcase (r=.4). This reduction of the relaxation parameter is due to stability

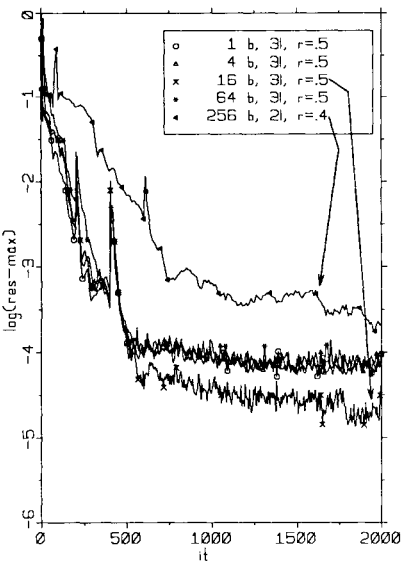


Figure 6. Influence of domain decomposition on maximum of residuum

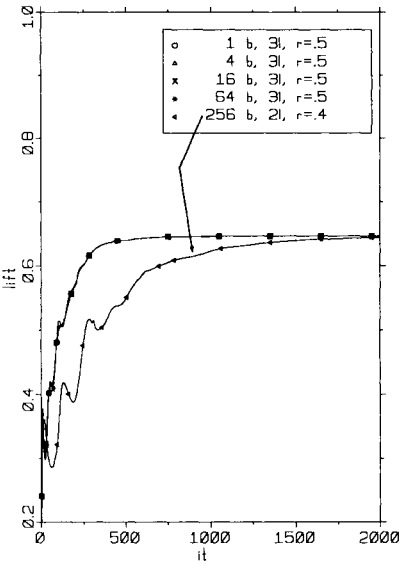


Figure 7. Influence of domain decompositon on lift coefficient

issues with the Gauss–Seidel relaxation (domain decomposition irritates the transport of information of the Gauss–Seidel scheme). Up to 64 blocks, the domain decomposition has almost no effect on convergence behavior. The 256 blocks testcase shows slower convergence due to the decreased relaxation parameter and reduced grid levels. These results indicate that the Gauss–Seidel relaxation is applicable also for massively domain decomposited grids, especially for realistic 3–D applications with many more grid points compared to the testcase.

Further algorithmic investigations with a division of the NACA0012 wing into 2048 blocks (blocksize: 2 x 4 volumes) have been performed. The most important information resulting from these investigations was to make sure an overall consistent formulation of all values at the blockfaces. Perturbations caused by inconsistent formulations at block faces reduce the accuracy of solutions for massively domain decomposited grids.

6. Speedup measurements

To test the efficiency of the parallelization of NSFLEX speedup measurements have been carried out on different parallel platforms using different communication interfaces. Measurements are available for an IBM SP2 (thin nodes) with 1, 2, 4 and 8 node processes and a NEC–Cenju 3 with 2, 4, 8, 16 and 32 node processes. The benchmark testcases have been performed at the GMD (St. Augustin) for $real * 4$ floating point calculations. The compiler options used are "-O -qhssngl -qarch=pwr2 -qtune=pwr2" on the IBM SP2 and "-O" on the NEC-Cenju 3.

Two different iteration schemes for the fine grid and the coarse grid of the NACA0012 testcase have been considered. The iteration schemes tested are a simple relaxation procedure with 100 iterations and a full multigrid scheme (FMG) using 3 grid levels with an overall number of 300 iterations.

The speedup as a criterion for the parallel performance is defined as

$$speedup := \frac{wall\ clock\ time\ (N = 1,2)}{wall\ clock\ time\ (N = n)}$$

in the following. N is the number of node processes (one block per node). All speedup measurements on the IMB SP2 are related to a monoblock sequential calculation (N=1) whereas the measurements on NEC–Cenju 3 are related to a two block parallel calculation (N=2). A monoblock calculation on the NEC–Cenju 3 could not be executed for the given testcase due to the main memory available.

The results for the speedup measured on IBM SP2 are given in figs. 8 and 9. The difference of the results are related to the different communication interfaces used. In fig. 9 the POE communication with the high performance switch (HPS) shows linear and superlinear speedup, whereas the results of fig. 8 based on PVM and the Ethernet interface show the influence of a "slow" communication. The increasing communication effort of the coarse grid calculation compared to the fine grid calculation is demonstrated especially in fig. 8 where the communication tasks are time consuming. This is also true for the FMG scheme compared to the relaxation technique.

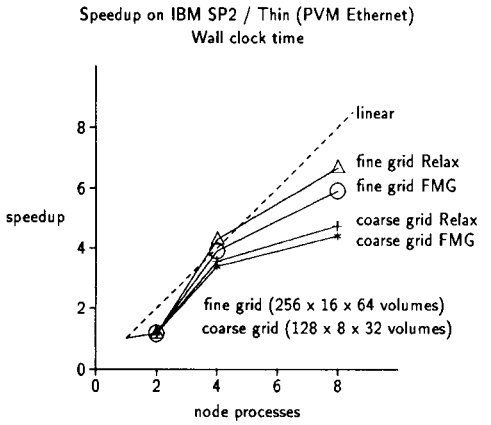


Figure 8. Speedup on IBM SP2, thin nodes – PVM, Ethernet

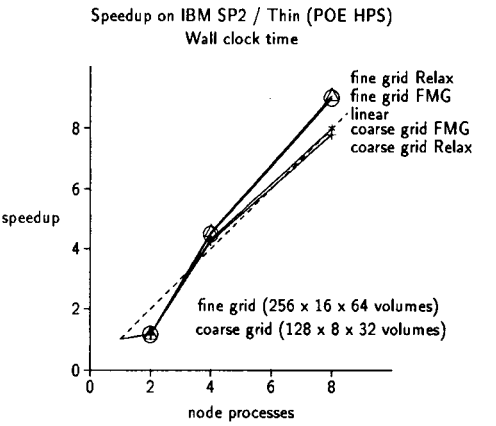


Figure 9. Speedup on IBM SP2, thin nodes – POE, HPS

The superlinear speedup of the fine grid calculations in fig. 9 is caused by cache effects. Especially the speedup for the 4 and 8 blocks testcase in fig. 9 demonstrates that the parallel implementation of the NSFLEX solver is a very efficient one. Only the 2 blocks testcase shows low speedup due to a cache problem. This cache problem also appears

in sequential reference calculations on thin and thick nodes. Fig. 10 shows the scaled efficiency (= 1./wall-clock-time; based on 1 block testcase, thin node) of the different testcases and different types of nodes. In contrast to the thin node, the thick node makes use of more cache and an additional 1MB second level cache. There is a great difference between the performance of the thick node and the performance of the thin node for the 2 blocks testcase only. The reasons are cache problems of the 2 blocks testcases that are reduced using the larger cache and the second level cache.

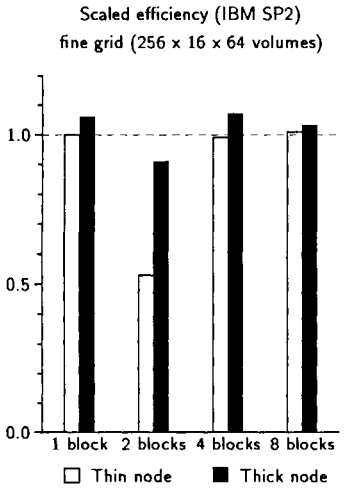


Figure 10. Scaled efficiency on IBM SP2, thin node versus thick node

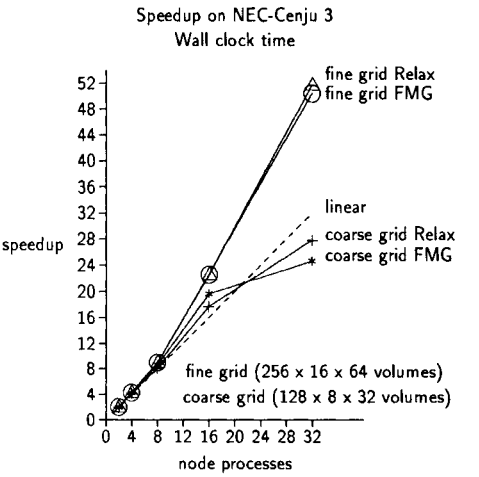


Figure 11. Speedup on NEC-Cenju 3

The results of the speedup measurements for the NEC-Cenju 3 are given in fig. 11. The coarse grid shows very good performance up to 32 blocks. The fine grid even indicates superlinear speedup that is related to the large cache available on this platform (32KB plus 1MB second level cache per node).

7. Results

The goal of the POPINDA project to develop a high level communications library that ensures easy and economical parallelization of industrial applied codes has been reached. The concept of parallelization ensures comfortable handling of the mastercodes. The parallel I/O, supported by the CLIC, results in a very simple host-program because there is no need to simulate the call sequence of the node-program on the host process. A parallel and a sequential version of the communications library combined with one single source of the mastercode enables easy manageable code development. A high degree of portability of the parallelized code is ensured by PARMACS calls and MPI based implementations of the communications library.

The algorithmic investigations for massively domain decomposited grids showed good convergence behavior of the Gauss–Seidel solver and indicated that fully consistent formulation of the algorithm at blockfaces is essential for accurate solutions. Benchmark testcases performed on IBM SP2 and NEC Cenju–3 demonstrate very good speedup. Using fast communication interfaces, superlinear speedup has been shown for gridsizes that are relevant for 3–D applications. The results also demonstrate that cache size can play an important role for speedup. The measurements for the PVM/Ethernet based calculations allow the conclusion, that the parallel NSFLEX implementation is also suitable for industrial applications on classical workstation clusters.

REFERENCES

1. Eisfeld, B.; Bleecke, H.-M.; Kroll, N. and Ritzdorf, H. : Parallelization of Block Structured Flow Solvers. *VKI special course on Parallel Computing in CFD*, von Karman Institute for Fluid Dynamics, Rhode-Saint-Genèse, 1995.
2. Schmatz, M. A. : NSFLEX – an implicit relaxation method for the Navier–Stokes equations for a wide range of mach numbers. In Wesseling, P., editor, *Numerical Treatment of the Navier–Stokes Equations*, pp. 109–123, Kiel, 1989. Vieweg. Proc. of the 5^{th} GAMM–Seminar, NNFM Vol. 30.
3. Schmatz, M. A. : Hypersonic three–dimensional Navier–Stokes calculations for equilibrium air. *AIAA–paper 89–2183*, 1989.
4. Schmatz, M. A.; Höld, R. K.; Monnoyer, F.; Mundt, Ch. and Wanie, K. M. : Numerical methods for aerodynamic design II. *Space Course Aachen, paper no. 62*, 1991. Also MBB–FE211–S–PUB–442.
5. Schmatz, M. A. : Three-dimensional viscous flow simulations using an implicit relaxation scheme. In Kordulla, W., editor, *Numerical simulation of compressible viscous-flow aerodynamics*, pp. 226–242. Vieweg, 1988. Notes on Numerical Fluid Mechanics, Vol. 22.
6. Eberle, A. : Characteristic flux averaging approach to the solution of Euler's equations. In *VKI lecture series 1987-04*, 1987.
7. Hänel, D.; Meinke, M. and Schröder, W. : Application of the Multigrid Method in Solutions of the Compressible Navier-Stokes Equations. In *Proc. of the fourth copper mountain conference on multigrid methods*, pp. 234–249, siam, Copper Mountain, Colorado, 1989.
8. Mundt, Ch.; Keraus, R. and Fischer, J. : New, accurate, vectorized approximations of state surfaces for the thermodynamic and transport properties of equilibrium air. *ZfW 15*, pp. 179–184, 1991.
9. Mundt, Ch.; Monnoyer, F. and Höld, R. K. : Computational Simulation of the Aerothermodynamik Characteristics for the Reentry of HERMES. *AIAA–paper 93–5069*, 1993. Also DASA–LME211–S–PUB–528.
10. Höld, R. K. and Fornasier, L. : Investigation of Thermal Loads on Hypersonic Vehicles with Emphasis on Surface Radiation Effects. *ICAS–paper 94–4.4.1*, 1994. Also DASA–LME12–S–PUB–543.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Towards an Integrated CFD System in a Parallel Environment

M. Shih*,
M. Stokes+,
D. Huddleston!,
B. Soni@

ABSTRACT

This paper describes the strategies applied in parallelization of CFD solvers in a parallel distributed memory environment using a message passing paradigm. To monitor the simulation process, indirect rendering via the OpenGL graphics API is investigated and shown to exihibit excellent performance. A customized, integrated CFD simulation system, TIGER, tailored for turbomachinery configurations, executable in a distributed environment is developed. computational examples demonstrating the successful parallelization of the NPARC code with real-time visualization and the TIGER system are presented. Future expansion to this environment is discussed.

1. INTRODUCTION

The primary objective of the National Science Foundation's Engineering Research Center (ERC) at Mississippi State University is to reduce the man/machine resource required to perform Computational Field Simulation(CFS). The approach to reducing the machine resource is two-fold: 1) design of specialized hardware such as multi-computers with very low latency communications, and 2) the design of distributed computing environments to take advantage of heterogeneous collections of personal computers, workstations, and supercomputers if available. The approach to reducing the manpower resource is primarily through the development of advanced software tools which efficiently generate and edit grids, and visualize the solution field.

Classically, the process of CFS is thought of as a pre-processing phase of grid generation, the solution phase, and a post-processing phase comprised of visualization. However, examination of this process in a production environment illustrates these phases are not separate at all, and typically exist in at least binary pairs. For example, grid generation requires visualization of the grids during the construction process to be effective. Grid generation requires one or more iterations of the field solution to adequately capture viscous phenomena and shocks. The solution process also requires visualization to determine the accuracy of boundary and run-time parameters. For more complicated problems, the solution phase may require an integrated grid generation capability to provide solution adaptivity.

To address this problem, the ERC formed the Integration Thrust to investigate the concept of integrated environments for the CFS process. Conceptually, an integrated environment would reduce or simplify the logistics of moving between phases and therefore reduce wall-clock time as well as

*. Post Doctorate, NSF Engineering Research Center for Computational Field Simulation
+. Research Faculty, NSF Engineering Research Center for Computational Field Simulation
!. Associate Professor, Civil Engineering, Mississippi State University
@ Professor, Aerospace Engineering, Mississippi State University

reduce the expertise required to perform simulations. Early testbeds for integrated systems revealed that simple–minded integration of existing grid generation, solver, and visualization tools were impractical, and would have limited longevity due the volatile nature of current day tools. These testbeds resulted in two conclusions which coincide with near–term and far–term goals, respectively:

1. The solver set–up/execution phase and real–time visualization can be integrated in a distributed environment using *off–the–shelf* technology.

2. Grid generation codes must be redesigned to support distributed memory environments and support object–oriented concepts and therefore are not currently mature enough for integration.

Concentration in this paper is placed upon the solver set–up/executation phase and the real–time visualization.

The casual observer may wonder why visualization must be distributed and why it should be real–time. The anwer to the first question is that current trends in CFS point to using a distributed memory parallel environment for large scale applications[1.]. With simulations routinely containing more than a million nodes, memory requirements for graphic workstations would overwhelm even the most current hardware, thus the distributed architecture is required to mass enough memory to contain the problem. The real–time features are required to monitor early or transient phases of the solution to minimize loss of computing resource due to incorrect boundary or run–time parameters. Note that what is needed here is not publication quality graphics, but rather graphics which illustrate faults in the solution process.

The progress realized in the development of the aforestated integrated environments for the CFS process is presented in this paper. In particular, the development associated with following issues are described:

i. Parallelizing the widely utilized off–the–shelf CFD system. The NPARC code[2.] was chosen for this research effort due to it's availability to US research and academic communities, and that it is heavily used in US Federal Labs and industry. Though the NPARC code has been parallelized by private companies such as Boeing[3.], no parallel version was available for general distribution as of the time of this writting. The NPARC code was a natural choose for parallelization because of its multi–block structured design. This aspect is discussed in detail in the following section.

ii. Real–time visualization in a a distributed environment. Here, an architecture for indirect rendering using OpenGL graphics library is described

iii. Customized integrated CFD simulation process for a class of configurations. An integrated system, TIGER, customized for turbomachinery application is described.

2. THE NPARC SOLVER

The NPARC solver started life as a derivative of the ARC[7] suite of codes from NASA Ames, but gained popularity after being heavily modified by Cooper[2] at Arnold Engineering Development Center(AEDC) in Tullahoma, TN. under the name PARC, or Propulsion ARC code. The NPARC code, a close derivative of PARC, solves either the Reynolds–averaged Navier–Stokes equations or the Euler equations, subject to user specification. NPARC uses a standard central difference algorithm in flux calculations, with Jameson[9] style artificial dissipation applied to maintain stability. NPARC has two–time integration algorithms available including Pulliam's[7] diagonalized Beam and Warming approximate factorization algorithm or a multi–stage Runge–Kutta integration. The diagonalized algorithm was applied herein. Among other characteristics, NPARC utilizes conservative metric differencing, offers local time–step options, and provides a generalized boundary treatment. The code can be applied in either a single or multi–block mode. This results in a code which is quite general and easy to apply to complex configurations. The NPARC multi–block code supports only one grid block in memory at a time, caching the other blocks to disk. The main loop would write out the current

block (if not the first time), read in and process the second block, and the cycle would repeat. Similarily, boundary conditions were handled by files which were subsequently read by adjacent blocks once in memory. This design made it quite simple to produce a parallel code from this original design.

The PANARC, or **P**arallel **N**ational **ARC** code, as coined at the ERC, took the original design and actuall simplified it. Block caching was eliminated such that each process only operated on one block, and the boundary condition routines replaced the previous I/O calls with PVM[10] send/receive calls. Boundary and run−time conditions are defined in a FORTRAN namelist file and were not altered from the original format. However, each process only reads what is pertinent for each respective block.

To simplify the run−time logistics, PANARC assumes each process can access the same set of input files, which under the UNIX Operating System, is accomplished by NFS. PVM was choosen for the interprocess communication library, over the then immature MPI[11] libary because MPI lacks support for dynamic process management. However, it is apparent that from commercial industry support among a large class of workstation and supercomputer vendors, MPI will be the preferred choice for the message passing interface of the future. Features lacking now are expected to be included in future versions of MPI.

Memory in the NPARC code is allocated in the main program as a single work array. The only change to support indirect rendering was to replace the DIMENSION statement with a C language call that allocates shared memory (see Figure 1). This change allows any other process (with proper permissions) to access the main memory used by the solver. It is through this mechanism that the visualization module gains access to the field variables.

3. ARCHITECTURE FOR INDIRECT RENDERING USING OPENGL

The premise for the design for this system is the concept of domain decomposition, which relies on the solution domain being broken down into (for this case) equal size domains, each domain being mapped to a remote computer. To minimize data transfer during the visualization phase, the rendering calls are issued on local data rather than transfering the data to the display host (which was not an option since the graphics workstation did not have enough memory to store the data in the first place). This concept relies on the Method of Extracts[12] which suggests that what is being visualized is a subset or *extraction* dimensionally lower than that of the original data. In terms of solution monitoring, this is almost always the case.

A design contraint was for the visualization scheme be as unobtrusive as possible to the simulation process as well as minimize code modification to the solver. In addition, the visualization capability must be attachable/detachable at any time during the simulation. The process toplogy that evolved is shown in Figure 1.

In this topology, visualization is facilitated as a separate process on each host. Communication between the solver and visualization process on each processor is through shared memory. Optional features in this topology(shown in dashed lines) are (1) the existence of a solver process on the display host (not recommended for large scale visualization), and (2) enabled message passing between a visualization client on the display host and the solver clients. The latter option may be convenient, but generally is not recommended in that it violates the requirement that the solver and visualization processes be independent.

In the current model, the visualization client running on the display host has the responsibity of creating the shared window, acting on mouse and button events(which are not shared), clearing the various buffers before rendering begins, setting the projection matrix, and synchronizing the swapping of the back display buffer. It may also have the responsibility of rendering information for a domain if the user has elected to use the display host as an active node in the simulation. The visualization clients on the remote processors render to the shared window on the display host through the multi−threaded X11 Server, not the visualization client running on the display host. This greatly simplifies the programming aspects of distributed applications.

Communications between the visualization clients are primarily one−way, that is flowing toward the display hosts. The OpenGL[] calls are tokenized on the remote clients and sent to the X11 Server running on the display host using the GLX extension provided by Silicon Graphics. Thus, the

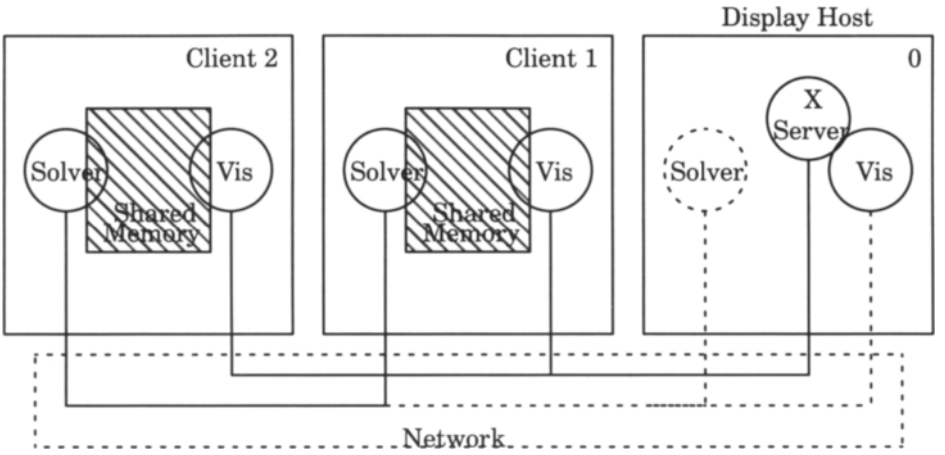


Figure 1. Process Topology of Indirect Rendering

programmer can think of indirect rendering simply as providing addition input streams into the geometry engines on the display host. Since this operation is often performed unsynchonized, the scan conversion may not be unique since these tokens may arrive unordered to the server. This is not generally a problem, because most operations are not dependent on the ordering. However, if alpha buffering(for example) is enabled with background blending, then the ordering of the operations are important and care must be taken.

It is often necessary to communicate information from the display host to the remote clients. Three different methods are described herein with the choice of methods dependent on the needs of the application. If the information is global in nature, then X Windows provides the mechanism known as properties, which are shared among local and remote processes sharing a common window. If any client attempts to change a global property, then a *PropertyChangeEvent* is sent to all clients, which respond with a request to read the new property and thus synchronize the data. If point−to−point communication is required, then a simple method is to post a window (such as Motif or Xt window) on the display processor where the user can use graphical techniques such as buttons or sliders to convey information. A third technique is to use client defined *XSendEvents* that support the passing of arbitrary information between local or remote processes. Care must be exercised when exchanging data between heterogenous machines, as data incompatibilities may arise. This problem can be avoided if the data is encoded/decoded using the publicly available XDR routines.

When double buffered graphics are required on the display host, it is the responsibility of the owner of the window(the visualization process on the display host) to actually swap the display buffers after the remote clients have completed their rendering operations. To perform this operation, the local client must have two pieces of information: (1) the number of remote clients sharing the window, and (2) notification when all clients have completed their operation. To satisfy (1), the visualization clients simply send a *ClientMessage* telling the client on the display host that a new remote client is registering itself. The local client simply keeps track of the number of registered clients. This information coupled with the XSynchronization Extension to X Windows allows the client on the display host to maintain a synchronized window.

4. TIGER SYSTEM

The development of TIGER has evolved from a grid generation system into an integrated system specialized in applications in rotating turbo−machinery. The integrated system is comprised of 6 modules: (*i*) grid generation module, (*ii*) visualization module, (*iii*) network module, (*iv*) flow solution module, (*v*) simulation module, and (*vi*) toolbox module. Each module may be accessed indepen-

dently to perform different tasks. These modules are linked together with a common graphical user interface (GUI).

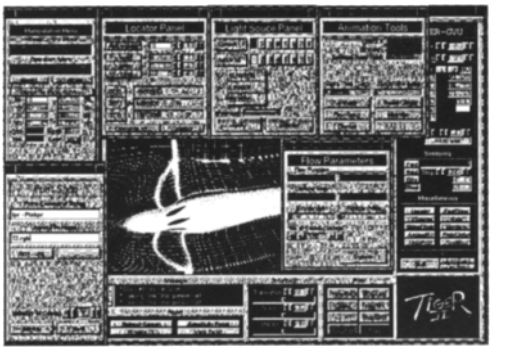


Figure 2. Graphical User Interface of TIGER

TIGER's GUI, as shown in Figure 2, was developed using the FORMS Library[14]. The interface contains a global panel window, an image display window, and various sub−panels and auxiliary panels. The global panel, where the logo, message browser, image controls widgets, and other global function buttons reside, is the main panel of the entire GUI. The image display window sits in the left−hand upper corner of the global panel and is the window that allows the user to view the grid and the solution. Sub−panels such as the "GVU" panel, pop up on the right hand portion of the global panel as the algorithm requests user inputs. These panels provide the necessary widgets for a particular group of user inputs. Auxiliary panels, such as the "Animation Tools" and "Flow Property" panels, pop up only upon the user's request for parameter customization. A help window, available upon request, consists of a large browser window to display the contents of the help files, and several buttons for the user to select topics.

The simulation capability in TIGER system is comprised of the network module, visualization module, and the flow solution module. As shown in Figure 3, the setup of this module is to use the network module as the data bridge that conveys user's commands and the flow solution between the local graphics workstation and a remote super computer. It consists of two sub−modules: the *client−server* sub−module, which resides in TIGER as a function, and the *server−client* sub−module which is a set of routines that reside independently on the remote mainframe. The network setup in the system allows two−way communication between the local workstation and the remote supercomputer, which allows the flow data to be sent back to local machine for the update. Figure 3 represents the setup of the network module. The setup of the simulation module is illustrated in Figure 4.

The flow solver that resides on the super computer may be any appropriate software. Currently, a multi−stage unsteady turbomachinery code (MSUTC)[15] is adopted as the flow solution module. This code applies the thin−layer approximation with a Baldwin−Lomax turbulence model to solve the Reynolds−averaged Navier−Stokes equations. It is an implicit finite volume scheme with flux Jacobians evaluated by flux−vector−splitting and residual flux by Roe's flux−difference−splitting. A modified two−pass matrix solver based on the Gauss−Siedel iteration was used to replace the standard LU factorization to enhance the code's stability. This code, uses localized grid distortion technique [15−16] to the buffer zone between the blade rows to achieve time−accurate solution for compressible flow. It performs flow calculation on the supercomputer, and outputs the flow solution every interval, whose length is specified by the user. The evolving solution is automatically updated on the screen graphically through the execution of visualization module. The visualization module is designed to allow scientific visualization for both the grid and flow solutions with basic rendering

methods, such as wireframe and Gouraud shading, contour and vector field. Time–dependent grid and/or solution can be animated dynamically.

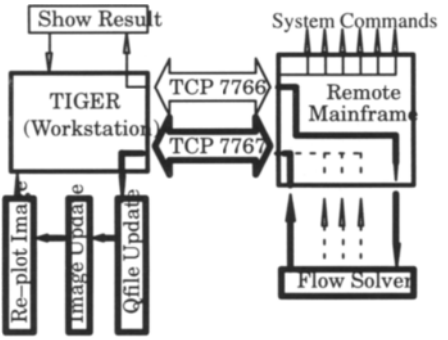


Figure 3. Tiger Network Topology

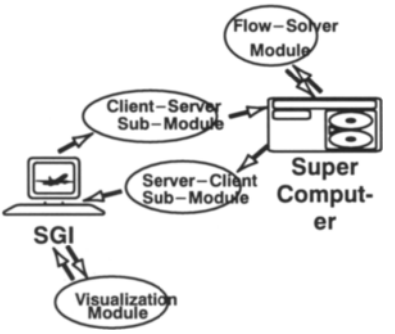


Figure 4. Simulation Module Setup

5. COMPUTATIONAL EXAMPLES

To test the design of the integration of the distributed parallel environment with distributed visualization, a four block 3D test case was performed using a four block 16mb SGI Indigo Elans workstations on a 10 mb Ethernet network(Figure 5 shows the block configuration on the vertical symmetry plane). The test case is an internal engine configuration with blades removed. Figure 6 is a six frame sequence of pressure on a vertical symmetry plane as captured directly from the screen.

For the given test cases, the latency created by network traffic is almost negligible. To further minimize thie effect, the X server on the display host is capable of caching display lists created on either the remote or local clients. The impact of caching is that objects that have not changed are not retransferred across the network, thus local redraws even for complex objects are very fast.

For the given test cases, the latency created by network traffic is almost negligible. To further minimize this effect, the X server on the display host is capable of caching display lists created on either the remote or local clients . The impact of this caching is that objects that have not changed are not retransfered across the network, thus local redraws even for complex objects are very fast.

Hamilton–Standard's single rotation propfan SR–7 is presented as an example simulation application. This geometry is modeled using a C–type domain with a grid size of 31x16x9, as illustrated in Figure 7. This geometry has 8 blades and the blade stagger angle is set to be 54.97 degrees, with an advance ratio J=3.07 (rpm=1840) and a free–stream Mach number 0.78. Each iteration of an Euler solution will take about 5 seconds on the aforementioned SGI Challenge machine. Density contours of the 2000th iteration is shown in Figure 8.

6. FUTURE WORK

The current effort clearly illustrates the need for an integrated desktop tool for the execution and field monitoring of CFS. This desktop, while generally not adding any new capability, aids the user by either automating functions, or simplifying the use of the tools by bringing them together in a common Graphical User Interface(GUI). A few of these tools are listed below.

Graphical based input/run parameters–
Currently, NPARC reads a FORTRAN namelist input file for information regarding the run–time and boundary conditions. A valuable aid to the user would be to allow these parameters to be edited graphically, and then have the GUI regenerate the namelist input files

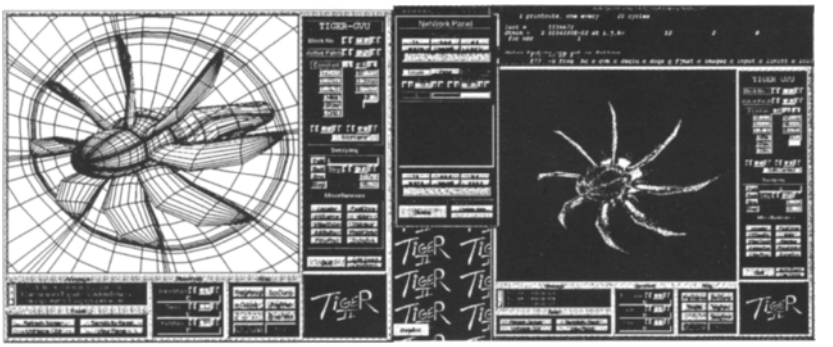


Figure 7. Grid for SR−7

Figure 8. Density contour for iteration 2000

for the user. The GUI would allow the user to indicate boundary conditions by *point−and−clip* thereby combining the editing process with graphical validation of the input.

An automatic mesh decomposition tool−
This feature, given a set of rules customizable by the user, would automatically decompose an arbitrary set of structured grid blocks into the number of equal sized grid blocks comensurate with the number of available processors in the distributed environment. This tool, given the connectivity information of the blocks, would be capable of breaking structured grids into sub−grids, including those that cross existing grid boundaries assuming the grid interfaces are point matched. This tool would also be capable of automatically calculating the boundary conditions on sub−grids.

Visualization monitoring tool−
This would allow the user to graphically monitor the state of a running simulation using the standard graphical features available in other post−processing packages such as line or garound shaded field properties, iso−surfaces, vector plots[5], streamlines, and particle paths[4.] to name a few. The interface would provide feature detection such as monitoring of flow angles on slip surfaces, or massive defects in total temperature or pressure, to name two. The ability to monitor global as well as local grid convergence at a glance would provide instant information on the *health* of the simulation.

Performance monitoring of the distributed parallel system−
This feature would allow the user to monitor the performance of the distributed system. Many of these packages already exist[5. ,6.], and would therefore only require merging into the integrated system.

As we look forward into the future of CFS as defined by the requirements for multi−disciplinary optimization and analysis(MDO&A), one can define requirements that must be present in grid generation systems to provide adequate functionality in an integrated system. These include

1. Ability to self−adapt to changes in either geometry or topology. Most current grid programs either don't support editing, or do it very poorly.
2. Closely related to 1, grid components must remember and maintain design constraints during the editing process. Without default editing methods, 1 would be undefined.
3. The inputs in the editing process must not be required to have intimate knowledge of the editing process itself. For example, if the location of a bezier control point attached to a line is altered and that line is connected to the edge of a surface, the procedure that requests the point to move need not be required to specify the series of steps necessary to reconstruct the surface. That information should be associated with the surface itself.

4. The editing process must not be linked uniquely to the GUI. Anyone or any process should be able to execute an edit, not just the indivual with the mouse.
5. Edits should occur in parallel.
6. Grids should be constructed in a collaborative environment. Many individuals should be able to work on components of a complicated grid simultaneously.
7. Grids components should be constructed relative to arbitrary defined sub−ordinate coordinate systems which are free to relocate in 3 space. This construct would be an extension of a grouping operator in which the geometic operators of translation, scaling, and rotation are defined.

8. REFERENCES

[1] Taylor, Steve, and Wang, Johnson, "Large−scale Simulation of the Delta II Launch Vehicle," *proceedings of Parallel CFD 95*, California Institute of Technology, June 26−28, 1995.

[2] Cooper, G.K., and Sirbough, J.R., "PARC Code: Theory and Usage," AEDC−TR−89.15, Arnold Engineering Development Center, Arnold AFB, 1989.

[3] Lewis, Jeffrey, "Domain−Decomposition Parallelization of the PARC Navier−Stokes Code," *proceedings of Parallel CFD 95*, California Institute of Technology, June 26−28, 1995.

[4] SHIH, M.H., "*TIGER: Turbomachinery Interactive Grid genERation*," Master's Thesis, Mississippi State University, December 1989.

[5] SHIH, M.H., "Towards a Comprehensive Computational Simulation System for Turbomachinery," Ph.D. Dissertation, Mississippi State University, May 1994.

[6] SHIH, M.H., YU, T−Y, and SONI, B.K., "Interactive Grid Generation and NURBS Applications", Accepted for print on Journal of Applied Mathematics and Computations, vol. 65:1−2, pp. 345−354, 1994.

[7] Pulliam, J.H., "Euler and Thin Layer Vanier−Stokes Codes: ARC2D, ARC3D", *Notes for Computational Fluid Dynamics User's Workshop*, The University of Tennessee Space Institute, Tullahoma, TN, UTSI Publication E02−4005−023−84, p. 151, March 1984.

[8] Cooper, G.R. and Sirbaugh, J.R., "The PARC Distinction: A Practical Flow Simulator", AIAA−90−2002, *AIAA/ASME/SAE/ASEE 26th Joint Propulsion Conference,* Orlando, FL, July 1990.

[9] Jameson, A., Schmidt, N. and Turkel, E., "Numerical Solutions of the Euler Equations by Finite Volume Methods Using Runge−Kutta Time−Stepping Schemer", AIAA−81−1259, *AIAA 14th Fluid and Plasma Dynamics Conference*, Palo Alto, CA, 1981.

[10] PVM reference: http://www.netlib.org/pvm3/

[11] MPI reference: http://www.mcs.anl.gov/mpi/

[12] STOKES, M.L., HUDDLESTON, D.H., and REMOTIQUE, M.G., "*A Proactical Model for Multidisciplinary Analysis Data and Algorithm*," 6th SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, VA, March 1993.

[13] SONI, B.K., THOMPSON, J. F., STOKES, M.L., and SHIH, M. H., "*GENIE^{++}, EAGLEVIWE and TIGER: General Purpose and Special Purpose Graphically Interactive Grid Systems*," AIAA−92−0071, AIAA 30th Aerospace Sciences Meeting, Reno, NV, January 1992.

[14] OVERMARS, M.H., "*Forms Library: A Graphical User Interface Toolkit for Silicon Graphics Workstations*," Version 2.1, Utrecht, Netherland, November 1992.

[15] CHEN, J.P. and WHITFIELD, D.L., *"Navier–Stokes Calculations for The Unsteady Flowfiled of Turbomachinery,"* AIAA–93–0676, AIAA 31st Aerospace Sciences Meeting, Reno, NV, January 1993.

[16] JANUS, J.M., *"Advanced 3–D CFD Algorithm for Turbomachinery,"* Ph.D. Dissertation, Mississippi State University, Mississippi, May 1989.

This Page Intentionally Left Blank

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel multi-block computation of incompressible flows for industrial applications

O. Byrde[a], D. Cobut[b], J.-D. Reymond[a], M.L. Sawley[a]

[a]Institut de Machines Hydrauliques et de Mécanique des Fluides,
Ecole Polytechnique Fédérale de Lausanne, CH–1015 Lausanne, Switzerland

[b]Département de Mécanique,
Université Catholique de Louvain, B–1348 Louvain-la-Neuve, Belgium

Numerical flow simulation for industrial applications necessitates considerable computational resources, not only for the resolution of the flow equations but also for the pre- and post-processing phases. We present the results of a study of the use of high-performance parallel computing to facilitate such numerical simulations. This study is being undertaken using a 256 processor Cray T3D system, within the framework of the joint Cray Research–EPFL Parallel Application Technology Program.

1. FLOW SOLVER

The parallel code used in this study is based on a multi-block code developed within IMHEF-EPFL for the numerical simulation of unsteady, turbulent, incompressible flows. This code solves the Reynolds-averaged Navier-Stokes equations on 3D structured and block-structured computational meshes.

1.1. Numerical scheme

The numerical method employs a cell-centered finite volume discretization with an artificial compressibility method to couple the pressure and velocity fields. A high-order upwind spatial discretization scheme based on the approximate Riemann solver of Roe is employed for the advection terms, while the diffusion terms are discretized using a central approximation. The time integration of the unsteady Navier-Stokes equations is performed using an implicit two-stage Runge-Kutta scheme. At each time step, a non-linear system resembling the equations for stationary flow is solved using the ADI method.

1.2. Parallelization

To date, only a subset of the original code has been parallelized, the parallel code can presently be used to compute steady, laminar, 3D incompressible flows. Two-dimensional flows are computed by replication in the third degenerate dimension. Parallelism is achieved by dividing the computational domain into a number of blocks (sub-domains), with the flow equations being resolved in all blocks in parallel by assigning one block to each processor. Communication between processors is necessary to exchange data at the edges of neighbouring blocks. The communication overhead is minimized by data localization using two layers of "ghost cells" surrounding each block. Data are exchanged between blocks using message passing (via the PVM library).

1.3. Optimization

The original code was written and optimized for vector supercomputers. To obtain optimal performance on the cache-based processors used in the parallel systems considered, most of the code has been re-written and hand optimized. While the Cray T3D offers a number of different parallel programming models (data parallel, message passing, work sharing, data sharing), message passing was chosen for the present code since a preliminary study [1] showed that it provides a good compromise between performance and portability for parallel multi-block flow solvers.

2. PERFORMANCE ANALYSIS

Studies have been undertaken to investigate the performance of the parallel code on the Cray T3D and, for comparison, on a cluster of 30 Hewlett Packard 9000/720 workstations interconnected via Ethernet. As a test case, inviscid flow between the blades of the "Durham low speed turbine cascade" [2] has been considered. Computations have been performed for both 2D and 3D flows using meshes containing 120x52x2 and 120x52x64 cells, respectively.

2.1. Comparison between the Cray T3D and a workstation cluster

Figure 1 shows the time required to perform a fixed number of iterations (4000) and the relative importance of the connectivity overhead (i.e. the time spent in communication and synchronization between blocks relative to the total time).

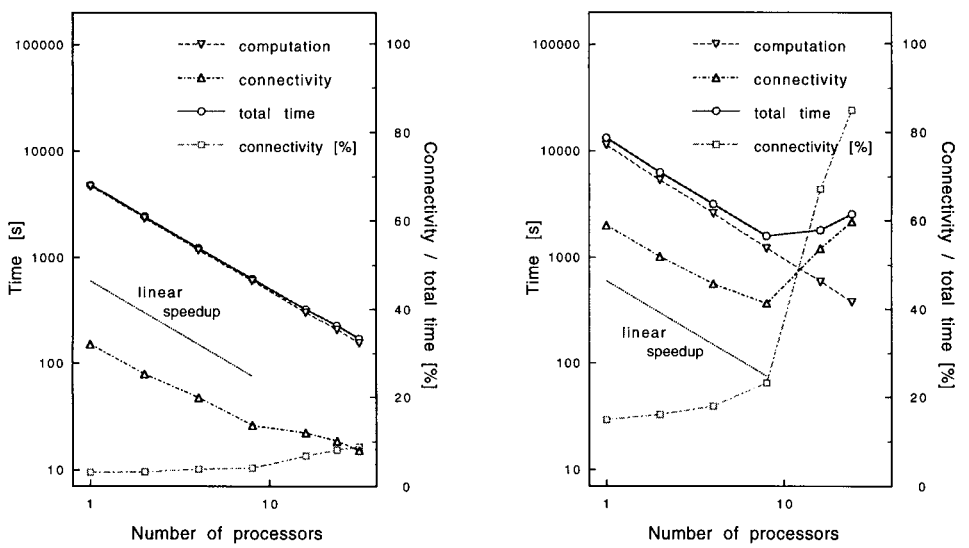


Figure 1. Wall clock time (left scale) and connectivity overhead expressed as a percentage (right scale) for computations performed on the Cray T3D (left) and workstation cluster (right).

Figure 1 shows that due to the very fast communication network of the Cray T3D, almost linear speedup is obtained. On the contrary, the performance of the workstation cluster decreases rapidly as the number of processors is increased, due to the low bandwidth of the Ethernet network. For example, using 16 workstations the connectivity overhead is about 58% of the total computation time, resulting in a time to solution longer than that necessary using 8 workstations!

2.2. Time to solution using a large number of processors

The present code uses an implicit numerical scheme to resolve the flow equations within each block combined with an explicit updating of the boundary values. It is therefore relevant to question if, due to convergence degradation, such a scheme is scalable to the large number of processors available on the Cray T3D system. Figure 2 presents the convergence history and time to solution (i.e. the wall clock time required to obtain a residual of 10^{-5}) as a function of the number of blocks employed.

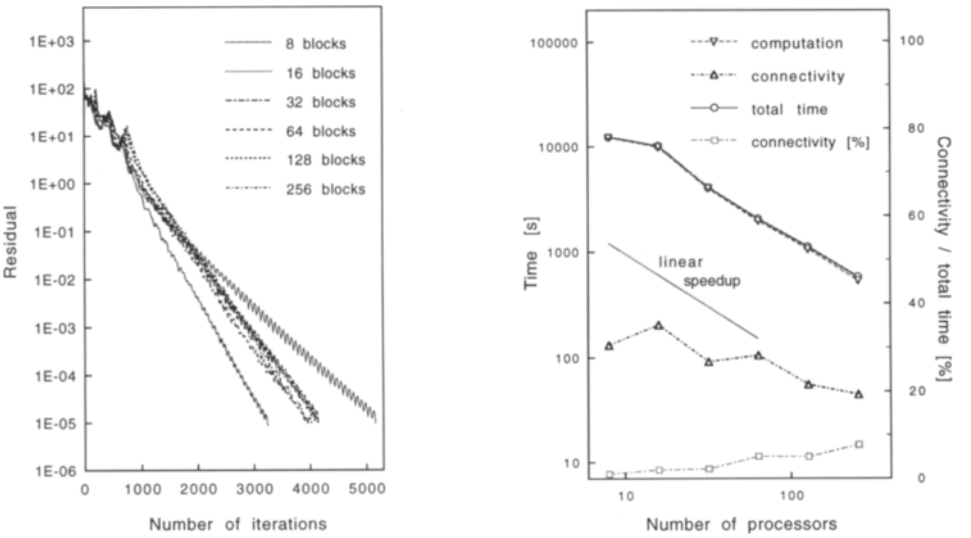


Figure 2. Convergence history (left) and time to solution (right) for 3D computations on the Cray T3D.

Figure 2 shows that the convergence degradation for the present test case is not significant, resulting in a nearly linear speedup with increasing number of processors. The observation that the slowest convergence is obtained using 16 blocks is presumably due to the fact that the convergence rate is determined not only by the number of blocks, but also by the transient solutions within each block. Figure 2 also shows that communication between processors of the Cray T3D is not significant for this test case even when a large number of processors is employed.

3. PRE- AND POST-PROCESSING

It is well recognized that for the numerical simulation of complex 3D industrial flows that involve an enormous quantity of data, the pre- and post-processing phases of the simulation procedure can necessitate a time (for data file manipulation and processing, I/O, etc.) often substantially longer than the cpu time required by the flow solver. This potential problem is further exacerbated if the flow solution is obtained using an efficient solver on a high-performance parallel system. For this reason, it is essential to incorporate the pre- and post-processing phases into the parallel environment.

3.1. Mesh generation

The construction of a suitable block structured mesh for complex 3D geometries comprises three stages: surface definition (generally employing CAD techniques); block boundary determination; and mesh generation within each block.

Block structured meshes, while being structured within each block, are generally irregular at the block level. The task of generating the block boundaries is therefore equivalent to determining an appropriate unstructured mesh. While the block boundary mesh is currently determined manually, it is desirable that its construction be performed using "automatic" unstructured hexahedral mesh techniques; such an approach is presently under investigation. Once the block structure is determined, the mesh generation within each block is undertaken in parallel, with the use of communication between processors to perform any necessary mesh smoothing across block boundaries.

The use of parallel mesh generation – even in the above-described limited form – has been shown to provide a substantial reduction in mesh generation time. In addition, since only the block boundary information is imported into the parallel system a significant reduction in I/O time is also obtained.

3.2. On-line visualization

Due to the enormous quantity of data generated by an unsteady 3D flow solver each time step, it is not practical to store these data on disk for later "off-line" processing and visualization. To overcome this problem, two "on-line" visualization procedures have been considered. The first is based on the PORTAL library [3] to interface to the AVS visualization software, while the second uses the IPC library to interface to the commercial TECPLOT visualization package (see Fig. 3). Both of these procedures use UNIX sockets (rather than message passing) for data transfer. Due to the significantly greater capabilities of TECPLOT, only the second approach has been retained.

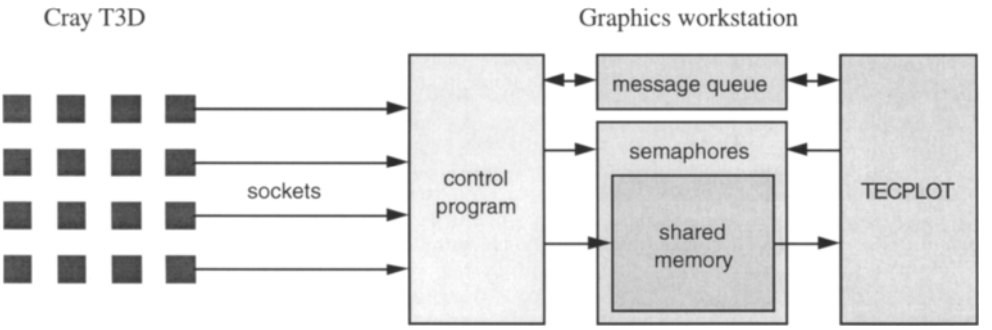


Figure 3. Schematic diagram of the on-line visualization procedure.

4. INDUSTRIAL APPLICATIONS

The parallel flow solver described above has been employed to compute a number of different large-scale 3D flows; two specific examples, of interest to the turbomachinery and automotive industries, are briefly presented in the following sections.

4.1. Water turbine

Water turbines of the Francis type are widely used for hydroelectric power generation. The numerical simulation of the flow in a Francis turbine is conventionally undertaken by computing the flow separately in each component of the turbine: spiral casing; distributor; runner and draft tube (see Fig. 4). The coupling of the flow in these components is complicated both by the

rotation of the runner relative to the other components and by the different periodicities of the distributor (24 blades for the present case) and the runner (13 blades).

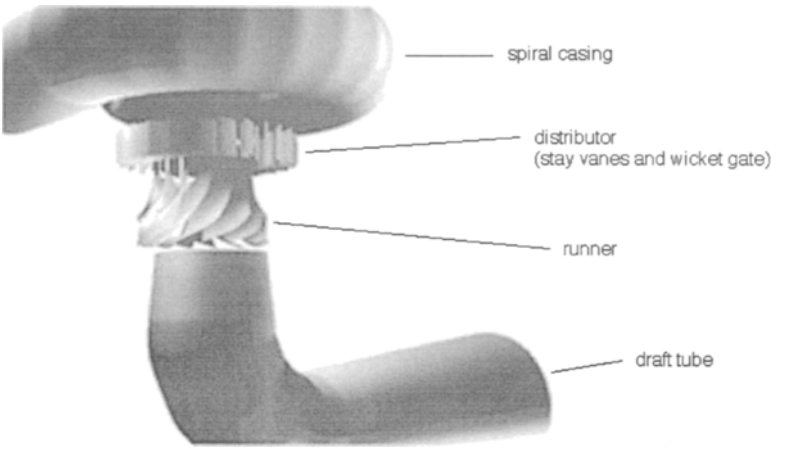


Figure 4. Exploded view of a Francis turbine

Most numerical simulations of the flow in the runner assume that the flow is periodic; this allows only one inter-blade channel to be computed rather than the whole runner. Using the computational power of the Cray T3D, the flow in the entire runner can be computed in a reasonable time, allowing an initial assessment of the flow coupling between the runner and the distributor. Figure 5 presents the results of a computation performed on 13 processors of the Cray T3D. For this computation, the periodicity of the flow at the exit of the distributor has been imposed as an input condition for the runner, resulting in non-periodic flow in the runner.

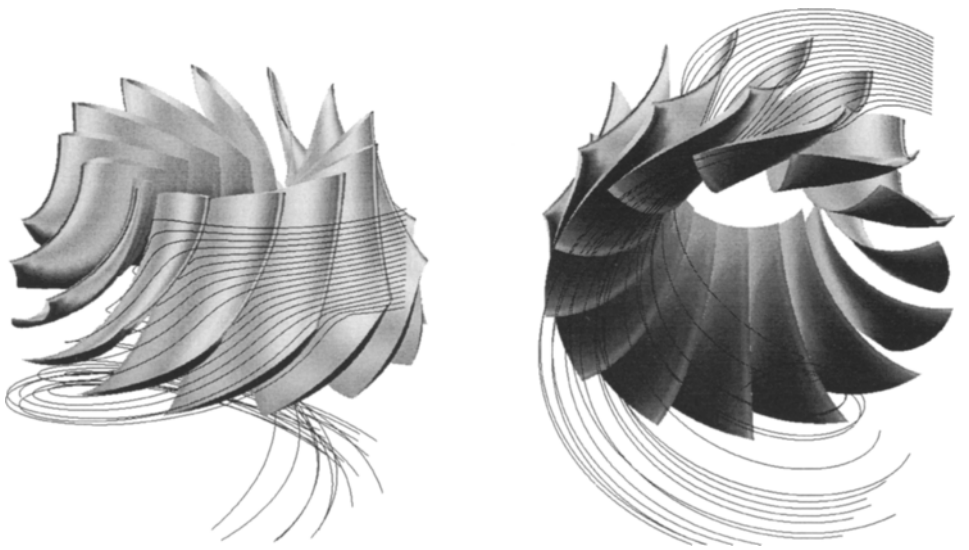


Figure 5. Surface pressure and streamlines for flow in the runner of a Francis turbine.

4.2. Formula 1 racing car

Due to the complex geometry and flowfield, it is currently not possible to simulate the flow around an entire Formula 1 racing car. The aerodynamic properties of the car are however principally determined by certain critical regions, such as the air inlet, undertray, wheels and front and rear wings. While the flow in these regions is coupled, it is nevertheless useful to study each region separately to obtain a detailed understanding of the flow. The present study has concentrated on a rear wing consisting of a series of multi-element airfoils (see Fig. 6). Four different configurations have been computed, corresponding to the three angles of the central element shown in Fig. 6 (denoted by c1-c3) plus the removal of this element (denoted by c0).

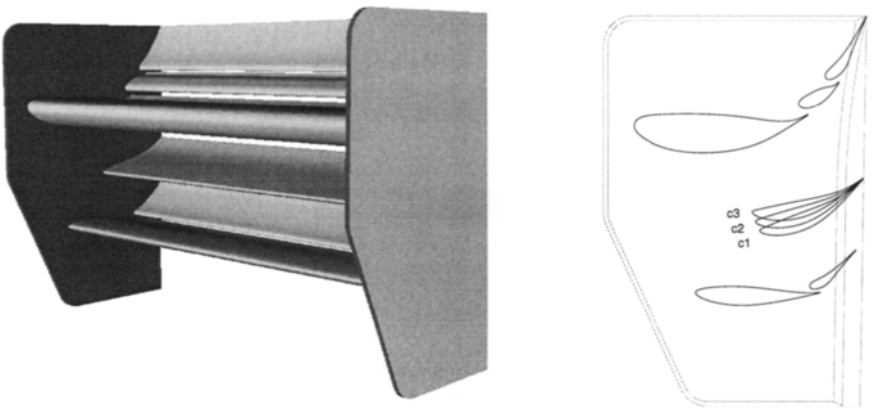


Figure 6. Geometry of the rear wing of a Formula 1 racing car.

The primary role of the rear wing is to produce downforce. Although significant drag also results, this is smaller than that produced by, for example, the wheels. Assuming that the flow is attached on the rear surface of each of the wing elements, inviscid computations should provide reasonable estimates of the downforce. It is also noted that even at maximum car speeds (up to 330 km/h), the flow can be considered to be incompressible.

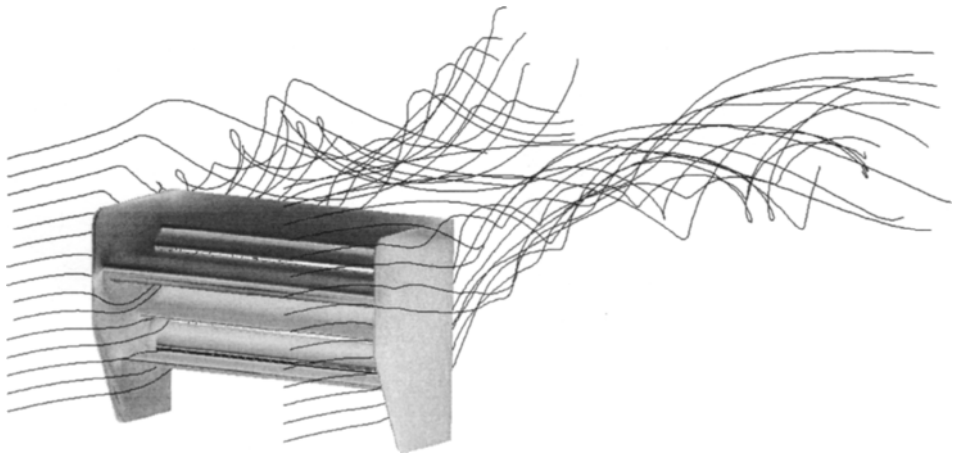


Figure 7. Surface pressure and streamlines for flow around the rear wing (configuration c2).

The numerical flow simulations performed to date have assumed zero angle of yaw; symmetry along the central vertical plane enables only half of the rear wing to be considered. For 3D flow computations, a mesh comprised of 121 blocks with a total of 3.8 million cells has been employed. Such computations required approximately 4 hours on 121 processors of the Cray T3D. The flow solution obtained for one of the rear wing configurations is presented in Fig. 7. These results clearly show the presence of outboard trailing vortices near the top of the side plates, as is observed on the race track.

Since the flow over a large portion of the wing elements is seen to be essentially two dimensional, computations have also been performed using a 2D mesh with 32 blocks. It should be noted that since there is not the same number of mesh cells in each block, load imbalance occurs, as shown in Fig. 8. This problem can be alleviated by computing more than one block per processor, as has been undertaken on the workstation cluster (see Fig. 8). Since the number of processors assigned to a job on the Cray T3D must be a power of two, the load balancing problem is less severe for the present flow case (as, in any case, some processors will be idle if the required number of processors is not a power of two). Nevertheless, it is planned to modify the parallel code to allow this procedure to be employed for more general computational meshes.

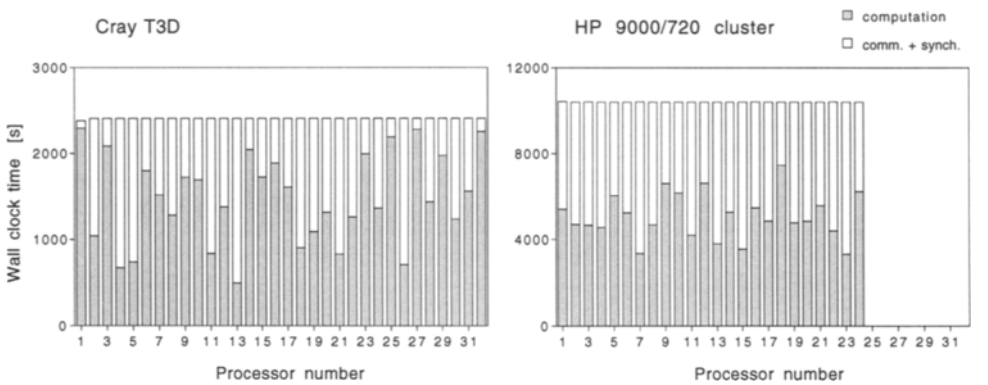


Figure 8. Wall clock time required for 2D flow computations of the flow around the rear wing.

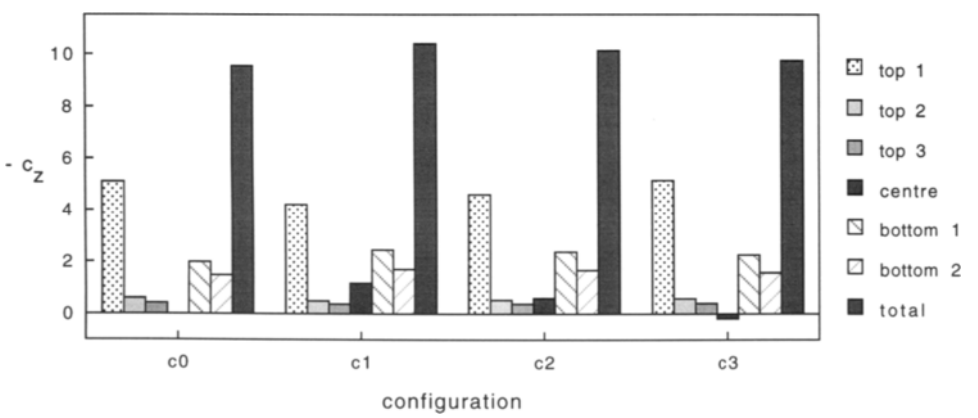


Figure 9. Component and total lift coefficient for the four different rear wing configurations considered.

A number of 2D flow simulations have been undertaken to investigate the characteristics of each wing configuration (in the symmetry plane). Computation of the pressure coefficient has allowed the contribution to the total wing downforce due to each element to be determined. The computed lift coefficients (normalized to the chord length of the largest element) are presented in Fig. 9 for each of the configurations considered. Figure 9 shows that a modification of the central element results in a change of the lift coefficient for each of the other elements, indicating that the flow around each component element is strongly coupled. For each configuration, approximately half of the total downforce is provided by the largest element.

5. CONCLUSION

The present study has shown that an existing 3D multi-block flow solver can be adapted to employ the enhanced capabilities of high-performance parallel computer systems. The use of the PVM message passing library has been seen to provide low communication overhead on the Cray T3D, as well as a high level of portability.

While workstation clusters can be considered as an alternative to supercomputers for small and medium size applications, the present study has demonstrated that large-scale applications can only be performed on massively parallel systems with the necessary memory requirements and computational power. The use of the Cray T3D system has enabled flow computations to be performed that are inaccessible to mono-processor computer systems.

The present study has reinforced the fact that to solve certain industrial problems, the computation time required by the flow solver is not the limiting factor, since the pre- and post-processing phases generally require significantly more time. It has been shown that the integration of the pre- and post-processing phases into the parallel environment can produce a significant reduction in the overall time to solution.

ACKNOWLEDGEMENTS

The authors wish to thank Y. Marx for the numerous discussions regarding the original code and S.A. Williams for his aid with visualization. The geometry of the rear wing of the Formula 1 racing car was provided by PP Sauber AG. Financial support was provided by the Cray Research–EPFL Parallel Application Technology Program. One of the authors (D.C.) received additional support via the European Project ERASMUS.

REFERENCES

1. M.L. Sawley and J.K. Tegnér, *A comparison of parallel programming models for multi-block flow computations*, accepted for publication in Journal of Computational Physics.

2. D.G. Gregory-Smith, *Test case 3: Durham low speed turbine cascade*, ERCOFTAC seminar and workshop on 3D turbomachinery flow prediction II (Val d'Isère, January 1994) Part III, pp. 96-109.

3. J.S. Rowlan and B.T. Wightman, *PORTAL: a communication library for run-time visualization of distributed, asynchronous data*, Proceedings of the Scalable High-Performance Computing Conference (Knoxville, May 1994) pp. 350-356.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel Benchmarks of Turbulence in Complex Geometries

Catherine H. Crawford, Constantinos Evangelinos
David Newman and George Em Karniadakis*

Center for Fluid Mechanics, Division of Applied Mathematics
Brown University
Providence, RI 02912

In this paper we present benchmark results from the parallel implementation of the 3-D Navier-Stokes solver *Prism* on different parallel platforms of current interest: IBM SP2 (all three types of processors), SGI Power Challenge XL, and the Cray J90/C90. The numerical method is based on mixed spectral element-Fourier expansions in $(x-y)$ and $z-$directions, respectively. Each (or a group) of Fourier modes can be computed on a separate processor as the linear contributions in Navier-Stokes equations are completely uncoupled. Coupling is obtained via the nonlinear contributions (advection terms) and requires a global transpose of the data and 1-D multiple-point FFTs. We define 3-D benchmark flow problems in prototype complex geometries, and measure parallel scalability and performance using different message passing libraries. Our results provide a measure of the sustained performance of individual processors in a parallel run, and indicate the limitations of the current communications software for typical problems in computational mechanics based on spectral or finite element discretizations.

1. INTRODUCTION

Direct Numerical Simulation (DNS) of turbulent flows refers to the method of computing flow fields resolving all scales from first principles without any *ad hoc* modeling assumptions. Since the first DNS of homogeneous turbulence in the early seventies computed on the CDC 7600, the field of turbulence simulation has developed in two directions [1]. First, realistic turbulence simulations are now regularly performed in simple geometries at wind-tunnel Reynolds numbers [2], [3]. While these simulations are now performed on parallel computers [4], [5], [6], vector supercomputing on the different Cray models has been primarily utilized in performing turbulence simulations in simple geometries [7], [8]. Second, simulations of turbulent flows in prototype complex geometries are now emerging; they are closely associated with the advances in parallel distributed memory computing [9]. Our interest lies in simulating flows of the second category.

In this paper we define two benchmark 3-D flow problems, representing both wall-bounded flows with rough surface as well as external flows past bluff bodies. These

*Author for correspondence

two flow problems represent a broad class of flows of physical interest and they provide an appropriate testbed for discretizations and parallel algorithms for complex geometry problems. Our discretization is based on mixed spectral element-Fourier expansions for geometries where one direction (z) is homogeneous, e.g. 3-D flow over a long circular cylinder. Fourier expansions are employed in the homogeneous direction while spectral element discretizations [10] are used on planes ($x-y$) normal to the homogeneous direction. This choice allows a Fourier decomposition of the Navier-Stokes equations, where all linear work associated with each Fourier mode is totally decoupled and as such it can be done in parallel; coupling comes through the nonlinear terms.

The overall algorithm, implemented in the code *Prism* [11], consists of two passes: The first pass treats the nonlinear terms by performing 1-D FFTs along the $z-$direction assuming that data is available as "pencils" on each processor. To proceed with the second pass, a global exchange of data must be performed so that data is available as "sheets", i.e. ($x-y$) planes, on each processor. We then solve for the 2-D Fourier coefficients on each plane associated with the corresponding Fourier mode. In this method, each processor will handle several pencils in the first pass and one (or more, memory permitting) Fourier modes in the second pass. For arbitrarily complex geometries, hexahedral or tetrahedral spectral elements should be employed in the discretization [12], [13] and different techniques, as described in [14], [15] can be used to distribute spectral elements among the processors.

We present 3-D results obtained on multiple processors following the aforementioned parallel algorithm. Moreover, we compare performance on the IBM SP2 [16] using the portable PVMe and MPI-F message passing libraries [17], [18], and we contrast these results with results obtained on a single node for the 2-D problem. Our main concern here is *scaled* parallel efficiency, which measures scalability of the parallel system as both the number of processors and the problem size grow proportionally. We also present results from turbulence simulations where additional flow statistics are calculated as part of "run-time" processing. In these simulations, we concern ourselves with the added computational and communications cost from added analysis which is needed for the understanding of complex turbulent flows.

The paper is organized as follows: In section 2, we describe the spectral element-Fourier method used in *Prism* as well as discuss the parallel nature and the communications involved in the algorithms. Benchmark results for 3-D code are given in section 3.1, where we also include a performance comparison for various message passing libraries on the IBM SP-2. Results from the benchmark turbulence simulations, with turbulence statistics routines added in to the base code, are described in section 3.2. Finally, we summarize our findings in section 4.

2. NUMERICAL METHODOLOGY

The details of the implementation of the spectral element-Fourier method used in *Prism* can be found in [11]; here we review parts of the method to demonstrate the parallel nature of the algorithm. Later in this section we describe computational issues associated with the high-order polynomial discretization; for now we concentrate on the Fourier aspects of the method, as this is the key to making the algorithm parallel. Within a domain Ω, the

fluid velocity $\boldsymbol{u}$ and the pressure p can be described by the incompressible Navier Stokes equations,

$$\left.\begin{array}{rcl} \frac{\partial \boldsymbol{u}}{\partial t} & = & -\nabla p + \nu \mathbf{L}(\boldsymbol{u}) + \mathbf{N}(\boldsymbol{u}) \quad in \quad \Omega \\ \nabla \cdot \boldsymbol{u} & = & 0 \end{array}\right\} \tag{1}$$

where

$$\mathbf{L}(\boldsymbol{u}) = \nabla^2 \boldsymbol{u} \tag{2a}$$

$$\mathbf{N}(\boldsymbol{u}) = -\frac{1}{2}\nabla(\boldsymbol{u} \cdot \boldsymbol{u}) + \boldsymbol{u} \times \nabla \times \boldsymbol{u}. \tag{2b}$$

The non-linear operator $\mathbf{N}(\boldsymbol{u})$ has been written in rotational form to minimize the number of derivative evaluations.

If we assume that the problem is periodic in the z-direction, we may use a Fourier expansion to describe the velocity and the pressure, e.g. for the velocity,

$$\boldsymbol{u}(x, y, z, t) = \sum_{m=0}^{M-1} \boldsymbol{u}_m(x, y, t) e^{i\beta m z} \tag{3}$$

where β is the z-direction wave number defined as $\beta = 2\pi / L_z$, and L_z is the length of the computational domain in the z-direction. We now take the Fourier transform of equation (1) to get the coefficient equation for each mode m of the expansion,

$$\frac{\partial \boldsymbol{u}_m}{\partial t} = -\tilde{\nabla} p_m + \nu \mathbf{L}_m(\boldsymbol{u}_m) + \boldsymbol{FFT}_m[\mathbf{N}(\boldsymbol{u})] \quad in \quad \Omega_m, \quad m = 0...M-1 \tag{4}$$

where $\boldsymbol{FFT}_m$ is the m^{th} component of the Fourier transform of the non-linear terms and,

$$\tilde{\nabla} = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, im\beta) \tag{5a}$$

$$\mathbf{L}_m(\boldsymbol{u}_m) = (\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} - \beta^2 m^2)\boldsymbol{u}_m. \tag{5b}$$

The computational domain Ω_m is an x-y slice of the domain Ω, implying that all Ω_m are identical copies. From equation (4) we see that the only coupling between modes is through the non-linear terms. Therefore the computation of each mode m can be done independently of one another. The obvious parallelization is to compute mode m on processor m for $m = 0...M-1$, so that the 3-D computation essentially becomes a set of M 2-D problems computed in parallel. The coupling of the non-linear term (Pass I) involves five steps:

1. Global transpose of velocities and vorticities.
2. N_{xy} 1-D inverse FFTs for each velocity and vorticity component,
   (where N_{xy} is the number of points in one x-y plane divided by the number of processors).
3. Computation of $\mathbf{N}(\boldsymbol{u})$.
4. N_{xy} 1-D FFTs for each non-linear term.
5. Global transpose of non-linear terms.

3. RESULTS

3.1. Three-Dimensional Benchmarks

3.1.1. Scalability

For the 3-D benchmarks we chose to test the absolute performance as well as measure the scalability of *Prism 3D*. We performed simulations using a mesh of K elements, with spectral order N and $N_z = 2M$ resolution in the z-direction on P processors. In Figure 1 we present results for the first 3-D benchmark problem, ie. cylinder flow at $Re = 200$ (with history point output for 10 points at every 10 timesteps). The problem size per processor (i.e., Fourier modes per processor) is kept constant by scaling the problem with the number of processors. Here $N = 9$ and direct solvers are used for both the pressure and the viscous step.

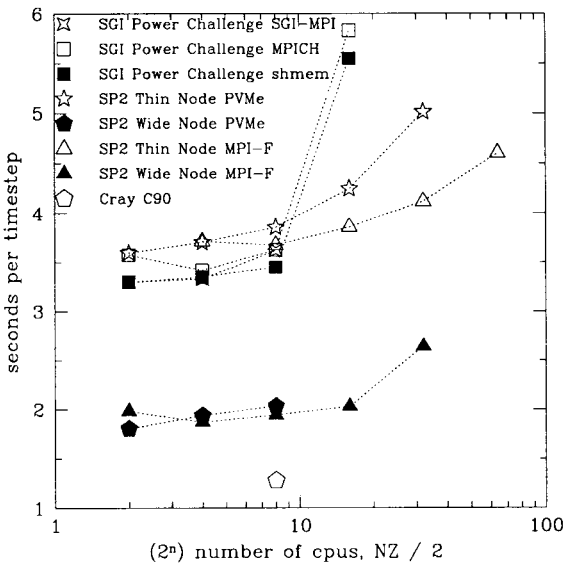


Figure 1. *Prism 3D* scalability. A 32 processor 66MHz Meiko CS-2 took 10.2s per timestep with a *projected* 5.6s per timestep for a 142MHz HyperSparc CS-2. The PVMe results were obtained using the fast memcpy with css_interrupts off for maximum performance.

For the simulations in Figure 1 the MPI and PVM version of *Prism 3D* were used along with a Cray macrotasking version [19], and a TCGMSG version using IRIX shared memory (SHMEM) specific calls for the message passing.

We make the following observations:

- The IBM SP2 Wide Node is the fastest, except for the Cray C90 which doesn't scale above 16 processors.

- Mflops performance on the SGI Power Challenge XL degrades significantly going from 8 to 16 processors. This is probably due to contention problems on the shared memory bus. It is possible that an SGI Power Array of multiple 4 or 8 processor Power Challenges connected via HIPPI would alleviate this problem, once specialized communication software reduces the very high latency of the HIPPI interconnect.

- The (still under development) SGI vesrion of MPI achieves, at least for a small number of processors, the same performance as the TCGMSG/SHMEM version.

- The IBM SP2 (especially the Wide Node) offers very good scalability. The relatively large increase of more than 0.5 seconds (30%) going from 16 to 32 processors is due to a few abnormally long timesteps that appeared in our simulations, thereby increasing the average.

3.1.2. Communications

The basic difference between the two- and 3-D simulations comes from added communication among processors. The largest message sizes for *Prism 3D* come from the global exchanges in the non-linear step which scale as $\left(\frac{N_z}{P}\right)\left(K\frac{N^2}{P}\right)$. The smallest message sizes come from history point analysis (described in more detail in the next section), in which the messages scale as $(\frac{N_z}{P})$. Lastly, the messages in the global addition operation (which can be efficiently implemented in a binary-tree form) during the outflow pressure boundary conditions (in cylinder flow simulations) scale in size as $(N \times P)$.

Given that the global exchange is the most important data communication in the code it is instructive to look at it in more detail: A straightforward implementation of a global exchange involves $(P-1)$ send-receive pairs per processor for a total of $P(P-1)$ pairs. This is the way our PVM version of *Prism 3D* was implemented in the absence of a PVM function that performs this communication. Assuming that in the best case there is no contention, the communication cost for each global exchange for a "flat" topology is:

$$t_{ge} = (P-1) * \left(t_{latency} + \frac{N_z K N^2}{P^2 * Bandwidth}\right), \tag{6}$$

where *Bandwidth* refers to the bidirectional (pair-exchange) bandwidth and $t_{latency}$ is the latency associated with a send-receive pair-exchange. Equation 6 shows that for constant problem size per processor (ie. $\frac{N_z}{P} K N^2$ =constant), t_{ge} will become latency dominated, scaling at best linearly with the number of processors.

3.1.3. Message Passing Libraries

The analysis above suggests that it is very important to choose an efficient message passing library that fully exploits the communication hardware features. Figure 2 shows comparisons between two message passing libraries in the IBM SP2 Wide Nodes, MPI-F and PVMe, for simulations of the $K = 320$ element $N = 11$ cylinder mesh. Both of these libraries use the optimized protocol for communication over the High-Performance Switch (HPS) as opposed to IP communications over the HPS or ethernet. In particular, two curves for PVMe are shown on each figure corresponding to standard PVMe and PVMe with a fast memory copy and switch css_interrupt turned off for maximum performance.

As can be seen from these plots, PVMe performs worse than MPI-F for all of the message sizes of interest ($P > 4$). PVMe performs relatively worse compared to MPI-F for the small messages that are used in the analysis and pressure steps. In fact, as the message size for the non-linear terms decreases as a function of P, the high latency that PVMe suffers from becomes more apparent.

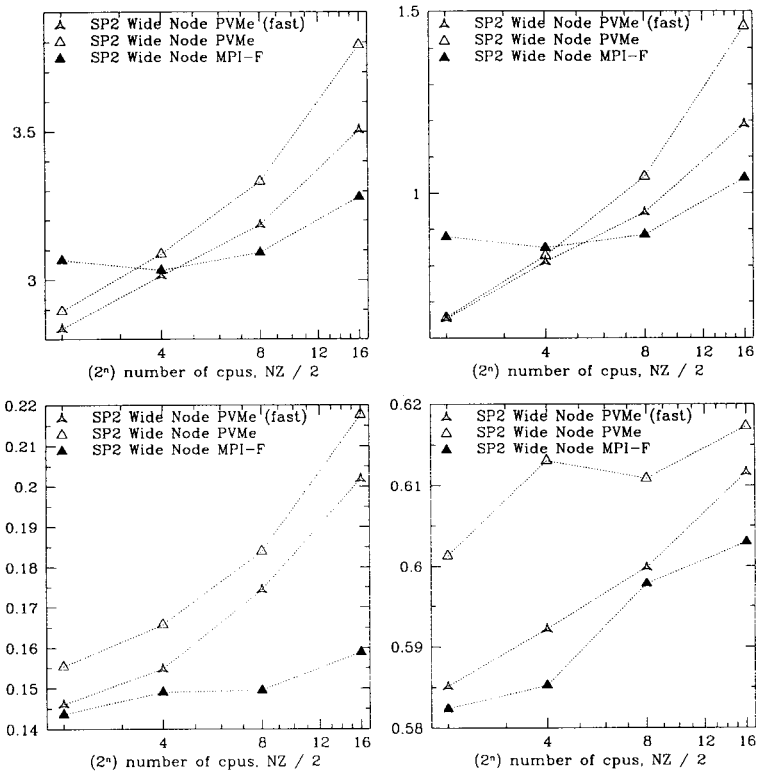


Figure 2. Results from the PVMe/MPI-F comparison done on the IBM SP2 Wide Nodes for $K = 320$ and $N = 11$. Clockwise from the top left is the time in seconds per timestep spent in the whole timestep, the non-linear step, the pressure step and the analysis step plotted versus the number of processors P. Medium size messages (of the order of tens of Kb) are used in the non-linear step, while small (of the order of tens and hundreds of bytes respectively) size messages are used in the analysis and pressure steps.

3.2. Turbulence Simulations

Direct Numerical Simulation is a useful tool for the study of turbulence from first principles. It can provide a wealth of information about the flow, from velocity time

traces, to forces on walls, to high-order vorticity statistics. Obtaining this information requires added computations, and for most parallel codes, added communication. We refer to this added computation and communication as *runtime* parallel data processing.

An example of added communication in collecting statistics without the added cost of communication is the Reynolds stress (τ_{Re}) analysis. Using Reynolds decomposition, we describe the total flow $\boldsymbol{u}$ and a mean $\overline{\boldsymbol{u}} = \boldsymbol{U}$ plus a perturbation $\boldsymbol{u}'$. The Reynolds stress is equal to an average of the product of velocity perturbations,

$$\tau_{Re} = -\overline{u'v'} = -(\overline{uv} - UV) \tag{7}$$

where u is the streamwise velocity and v is the normal velocity. To compute τ_{Re} the velocity products are computed in physical space (in parallel) during the non-linear step, where the velocities have already been transformed. A running sum of the appropriate quantities over the duration of simulation is kept.

Quadrant analysis, a type of statistical analysis of the velocity and vorticity, is one way to study turbulence structures. The analysis itself is done in post-processing, however, time histories in the flow must be obtained throughout the simulation. In *Prism 3D*, collecting time histories involves a series of small message sizes (of $\mathcal{O}(N_z/P)$) sent amongst the processors during the "Analyze" step in the code. Fourier coefficients are collected from all the processors for the given (x,y) gridpoint for all z-planes. Processor 0 executes a real FFT to get a "pencil" of physical space data and then searches this data set for the appropriate z coordinate. Statistics, which are even more difficult to efficiently compute in parallel, are velocity space correlation functions where the computations may be easily done in physical space, but the communication among processors to find the data of the two gridpoints in question can require extensive message passing time.

In order to see how code performance was affected by collecting various turbulence statistics, we chose two prototype complex geometry flows for benchmarking. The first was a continuation of our *Prism 3D* benchmarking – the $K = 320$ element cylinder mesh, this time using $N = 11$, $N_z = 32, (P = 16)$, and $Re = 500$. The second benchmark run was a turbulent channel with its lower wall mounted with riblets. The mesh used $K = 240$ elements. Runs were done using $N = 11$, $N_z = 64, (P = 32)$, at $Re = 3280$ (based on channel half-height and equivalent centerline laminar velocity). All results shown are from simulations on the IBM SP2 Thin Nodes using MPI-F.

In Table 1 we show the results for the cylinder benchmark with the added routines for turbulence data (*Wakes 3D*) as compared to *Prism 3D*. The statistics that were collected were as follows,

1. 10 velocity and pressure history points (4 fields),
2. kinetic energy per Fourier mode,
3. forces (lift and drag) on the cylinder,
4. velocity, vorticity, and τ_{Re} statistics.

Collecting the turbulent statistics added 0.73 seconds per timestep for the cylinder runs. In fact, 90% of this added cost is from the calculation of the velocity, vorticity, and τ_{Re}

Step	Prism 3D (sec/Δt)	Wakes 3D (sec/Δt)
Non-Linear	1.41	2.08
Pressure	1.39	1.37
Linear	3.36	3.40
Analysis	0.30	0.35
Total	6.47	7.20

Table 1
Timing data for the turbulent cylinder benchmarks at $Re = 500$. Times shown are wall clock times.

Step	Prism 3D (sec/Δt)	Rib 3D(sec/Δt)
Non-Linear	1.05	1.69
Pressure	1.08	1.11
Linear	2.29	2.34
Analysis	0.23	0.95
Total	4.60	6.08

Table 2
Timing data for the turbulent riblet channel benchmarks at $Re = 3280$. Times shown are wall clock times.

statistics, 6% is from the added communication in the history point analysis, and 4% is from the forces and kinetic energy computation.

The results from the riblet channel simulations from the turbulent statistics code, *Rib 3D*, as compared to *Prism 3D* are shown in table 2. For these simulations, the following statistics were computed:

1. 16 history points of velocity, vorticity, and pressure history points (7 fields),
2. 5 vorticity flux history points,
3. kinetic energy per Fourier mode,
4. drag force on the walls,
5. velocity, vorticity, velocity-vorticity transport, and τ_{Re} statistics.

In these simulations, the addition of turbulence statistics to the code have added 1.48 seconds per timestep, over 30% of the original total time per timestep. Unlike the cylinder runs where the added time was mostly from the added computation of the velocity, etc. statistics, the added time to the riblet runs is evenly distibuted between added *computation* in the statistics (43%) and the added *communication* in the history point analysis

(30%). The vorticiy flux analysis, which requires both added computation and communication, is responsible for 21.5% of the added time. The kinetic energy computation represents 5% of the additional time while the drag calculations are less than 1% of the turbulence statistics cost.

4. SUMMARY AND CONCLUSIONS

We have used a hybrid spectral element-Fourier code to simulate 3-D prototype flows in non-trivial computational domains. The computational complexity of this algorithm is typical of a broader class of algorithms used in computational fluid dynamics and general computational mechanics. Based on our results we can make the following observations:

In the benchmarks we performed on the parallel systems using the 3-D Navier-Stokes solver *Prism 3D* we evaluated absolute performance and scalability and their dependence on the choice of message passing library. In particular, we found that there is significant memory bus contention in shared memory systems for a number of processors more than about 10 (e.g. P=8 for the SGI Power Challenge XL). Scalability as measured by the scaled parallel efficiency is high for distributed memory systems. The selection of message passing libraries is important both for portability as well as for achieving high absolute performance. For example, in comparing PVM versus MPI on the IBM SP2 we found a significant deterioration of performance for PVM if a large number of small messages is involved across many processors. In general, the newer generation of RISC-based systems achieve comparable performance with vector parallel system but at lower cost, as we have seen with direct comparisosns with the parallel Cray C90.

Finally, production computing and in particular turbulence simulation, involves runtime global data processing for computing statistics, correlations, time-histories, etc. These operations typically involve several hundreds of very small messages among processors which may not be nearest neighbors as the computational algorithm requires. It is our experience that this may add significantly to the wall clock per time step. This added cost is usually overlooked but it is perhaps one of the most significant components as *parallel* simulations of turbulence become the prevailing mode of analysing turbulence numerically.

Acknowledgements

These benchmarks were performed on the Intel Paragon at the San Diego Supercomputer center, the IBM SP2 at the Maui High Performance Computing Center and the Cornell Theory Center, the SGI Power Challenge XL at the National Center for Supercomputing Applications and at SGI's Hudson offices, and the CRAY Y/MP C-90 at the Pittsburgh Supercomputing Center and at CRI's headquarters. We are greatful to the consultants at all of the centers for their help and expertise. We are also greatful to Ronald D. Henderson of Caltech for help in the original parallel port of these codes, Torgny Faxen of Silicon Graphics for TCGMSG/SHMEM port to the Power Challenge and his results using the SGI produced MPI library, David Daniel and Larry Watts of Meiko for their results on the CS-2, Paul Hastings of Cray Research for his parallel port of *Prism* to the Cray C-90 using macrotasking and Suga Sugavanam, Raj Panda, Fred Gustavson and Prasad Palkar of IBM for their help with IBM's ESSL mathematical subroutine library and the runs on the SP2 Thin Node 2.

This work is supported by DOE, AFOSR, ONR and NSF.

REFERENCES

1. Karniadakis G.E. and Orszag S.A. Modes, nodes, and flow codes. *Physics Today*, page 35, March 1993.
2. She Z.S., Jackson E., and Orszag S.A. Intermittent vortex structures in homogeneous isotropic turbulence. *Nature*, 344:226, 1990.
3. Tanaka M. and Kida S. Characterization of vortex tubes and sheets. *Physics of Fluids*, 5 (9):2079, 1993.
4. Pelz R.B. The parallel Fourier pseudospectral method. *J. Comp. Phys.*, 92 (2):296, 1991.
5. Jackson E., She Z.S., and Orszag S.A. A case study in parallel computing: I. Homogeneous turbulence on a hypercube. *J. Sci. Comp.*, 6 (1):27, 1991.
6. Wray A.A. and Rogallo R.S. Simulation of turbulence on the Intel Gamma and Delta. Technical Report NASA Technical Memorandum, Nasa Ames Research Center, Moffet Field, California, 94035, 1992.
7. Kerr R. M. Higher order derivative correlation and the alignment of small-scale structures in isotropic numerical turbulence. *Journal of Fluid Mechanics*, 153:31–58, 1985.
8. Kim J., Moin P., and Moser R. Turbulence statistics in fully developed channel flow at low Reynolds number. *Journal of Fluid Mechanics*, 177:133, 1987.
9. Chu D., Henderson R.D., and G.E. Karniadakis. Parallel spectral element-fourier simulations of turbulent flow over riblet mounted surfaces. *Theor. Comput. Fluid Dynamics*, 3:219–229, 1992.
10. Patera A.T. A spectral element method for Fluid Dynamics; Laminar flow in a channel expansion. *Journal of Computational Physics*, 54:468–488, 1984.
11. Henderson R.D. and Karniadakis G.E. Unstructured spectral element methods for simulation of turbulent flows. *Journal of Computational Physics*, to appear, 1995.
12. Maday Y. and Patera A.T. Spectral element methods for the incompressible Navier-Stokes equations. *ASME, State-Of-The-Art Surveys*, Chapter 3, Book No. H00410:71, 1989.
13. Sherwin S.J and Karniadakis G.E. Tetrahedral hp finite elements: Algorithms and flow simulations. *Journal of Computational Physics*, to appear, 1995.
14. Fischer P.F. and Patera A.T. Parallel simulation of viscous incompressible flows. *Ann. Rev. of Fluid Mech.*, 26:483–527, 1994.
15. Ma H. Parallel computation with the spectral element method. In *Parallel CFD '95, Caltech, June 26-28*, 1995.
16. *IBM Systems Journal*, 34(2), 1995.
17. Franke H., Wu E.C., Riviere M., Pattnaik P., and Snir M. MPI Programming environment for IBM SP1/SP2. Technical report, IBM T. J. Watson Research Center, P.O. 218, Yorktown Heights, NY 10598, 1995.
18. IBM, http://ibm.tc.cornell.edu/ibm/pps/doc/pvme.html. *Parallel Virtual Machine enhanced for the IBM SP2.*
19. Hastings Paul. Porting Prism 3D to the Cray C90. Technical report, Cray Research, Inc., 1994.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel Computation of 3-D Small-Scale Turbulence Via Additive Turbulent Decomposition

S. Mukerji and J. M. McDonough*

Department of Mechanical Engineering
University of Kentucky, Lexington, KY 40506-0108, USA

Implementation and parallelization of additive turbulent decomposition is described for the small-scale incompressible Navier-Stokes equations in 3-D generalized coordinates applied to the problem of turbulent jet flow. It is shown that the method is capable of producing high-resolution local results, and that it exhibits a high degree of parallelizability. Results are presented for both distributed- and shared-memory architectures, and speedups are essentially linear with number of processors in both cases.

1. INTRODUCTION

Turbulent flows have been investigated both theoretically and experimentally for over a century, but in recent years the wide availability of supercomputers has spurred interest in numerical techniques. However most of the currently used computational methods have deficiencies and limitations. The drawbacks of modeling approaches like mixing length and κ-ϵ are well known, see for example [3]. Subgrid-scale modeling continues to be a weak part of the large eddy simulation (LES) technique (cf. Ferziger[1]), even when dynamic subgrid-scale models (Germano et al. [2]) are used. Direct numerical simulation (DNS) of turbulent flows is restricted to flows with low Reynolds numbers (Re) because of limitations of computing hardware; thus simulation of flow problems of engineering interest ($Re > 10^6$) is not presently feasible (see Reynolds [9]). Moreover, DNS has a limited scope for parallelization which means that it cannot fully utilize the opportunities offered by massively parallel processors (MPPs) which present the only hope of solving realistic turbulent flow problems in the near future.

The technique used for turbulent flow calculations in the present research is the additive turbulent decomposition (ATD) first proposed by McDonough et al. [7] and developed by McDonough and Bywater [4,5] in the context of Burgers' equation, and by Yang and McDonough [10] for the 2-D Navier-Stokes (N.-S.) equations. The algorithmic structure of ATD is similar to LES; but it uses unaveraged equations, and hence there is no closure problem. Like DNS, it maintains full consistency with the N.-S. equations. Moreover, ATD is designed to fully exploit the architecture of the MPPs and offers the possibility of both fine- and coarse-grained parallelization. The work presented here represents the first study of ATD in three space dimensions, and parallelization thereof. The results indicate

*This research is supported by the Department of Energy under Grant No. DE-FG22-93PC93210.

that theoretical speedups predicted in McDonough and Wang [6] and verified there with one-dimensional calculations are, in fact, achievable in three dimensions. In particular, it is shown in [6] that if linear speedups can be achieved on the three main parallelization levels of ATD (see Figure 2), then effective run times in turbulence simulations can be reduced to at most $\mathcal{O}(Re^{5/3})$ from the usual $\mathcal{O}(Re^3)$; our preliminary results indicate that required linear speedups are possible.

2. ADDITIVE TURBULENT DECOMPOSITION

2.1. Procedure

To demonstrate the details of the ATD algorithm it will be applied to the viscous, incompressible N.-S. equations which are given in vector form as:

$$\nabla \cdot \boldsymbol{U} = 0\,, \tag{1}$$

$$\boldsymbol{U}_t + \boldsymbol{U} \cdot \nabla \boldsymbol{U} = -\nabla P + \frac{1}{Re}\Delta \boldsymbol{U}\,. \tag{2}$$

The above equations are non-dimensional where $\boldsymbol{U}$ is the velocity vector scaled with respect to U_0, a characteristic velocity scale, and P is pressure non-dimensionalized by ρU_0^2 (ρ is density). Re is Reynolds number defined as $U_0 L/\nu$ where L is a length scale, and ν is kinematic viscosity. The gradient operator and the Laplacian are denoted by ∇ and Δ respectively. The subscript t denotes the time derivative in the transport equation.

The dependent variables are split into a *large-scale* $(\bar{\boldsymbol{U}}, \bar{P})$ and a *small-scale* $(\boldsymbol{U}^*, P^*)$ as follows:

$$\boldsymbol{U} = \bar{\boldsymbol{U}} + \boldsymbol{U}^* \quad \text{and} \quad P = \bar{P} + P^*\,.$$

The large-scale quantities can be viewed as the first few modes in a Fourier representation of the total quantity, and the small-scale quantities are the series remainders consisting of high mode numbers. The split quantities are then substituted into the governing equations which take the form:

$$\nabla \cdot (\bar{\boldsymbol{U}} + \boldsymbol{U}^*) = 0\,, \tag{3}$$

$$(\bar{\boldsymbol{U}}_t + \boldsymbol{U}_t^*) + (\bar{\boldsymbol{U}} + \boldsymbol{U}^*) \cdot \nabla(\bar{\boldsymbol{U}} + \boldsymbol{U}^*) = -\nabla(\bar{P} + P^*) + \frac{1}{Re}\Delta(\bar{\boldsymbol{U}} + \boldsymbol{U}^*)\,. \tag{4}$$

The governing equations are now additively decomposed (in a manner analogous to operator splitting schemes) into large-scale and small-scale equations:

$$\nabla \cdot \bar{\boldsymbol{U}} = 0\,,$$
$$\bar{\boldsymbol{U}}_t + (\bar{\boldsymbol{U}} + \boldsymbol{U}^*) \cdot \nabla \bar{\boldsymbol{U}} = -\nabla \bar{P} + \frac{1}{Re}\Delta \bar{\boldsymbol{U}}\,, \qquad \text{(large-scale)} \tag{5}$$

$$\nabla \cdot \boldsymbol{U}^* = 0\,,$$
$$\boldsymbol{U}_t^* + (\bar{\boldsymbol{U}} + \boldsymbol{U}^*) \cdot \nabla \boldsymbol{U}^* = -\nabla P^* + \frac{1}{Re}\Delta \boldsymbol{U}^*\,. \qquad \text{(small-scale)} \tag{6}$$

It is clear from equations (5-6) that there are enough equations for the number of unknowns; that is, there is no closure problem. The decomposition is not unique, but the

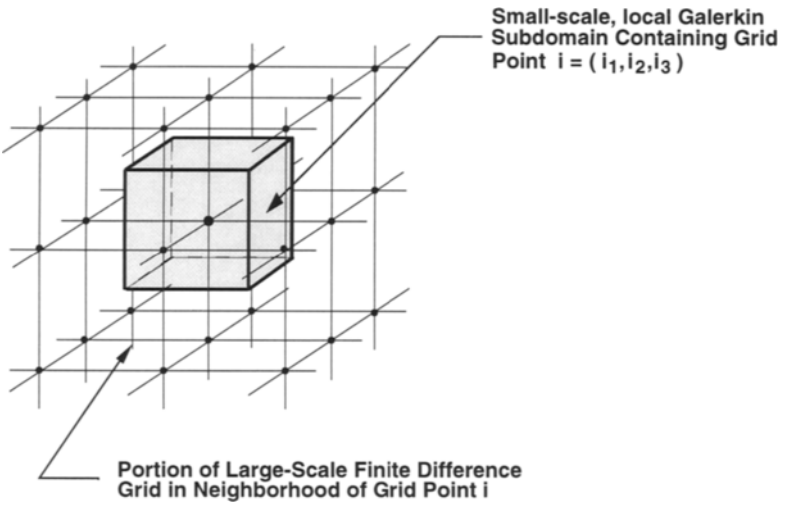


Figure 1. Domain decomposition employed in ATD

given form retains the structure of the N.-S. equations except for the additional *cross-terms*, e.g. $(\bar{\boldsymbol{U}} \cdot \nabla \boldsymbol{U}^*)$, which arises from decomposition of the nonlinear term of the original equation. The cross-terms also maintain coupling between the scales. Although at present the ATD algorithm is implemented with this two-level splitting, it could conceivably involve multi-level decomposition.

It should be noted that the consistency of the split equations with the N.-S. equations implies Galilean invariance. Also, realizability is automatically achieved because turbulent fluctuating quantities are calculated directly. The splitting also imparts the flexibility of using different numerical methods on each scale. Typically, finite difference (or finite volume) schemes are used for the large-scale which usually does not require high resolution, and Galerkin/spectral methods are used on the small-scale.

2.2. Parallelizability of ATD

Figure 1 shows a typical small-scale subdomain around a large-scale grid point. The small-scale equations are solved locally within this subdomain. The wide scope for parallelization in ATD is inherent in this spatial domain decomposition. Since there is a small-scale subdomain corresponding to each large-scale grid point, the small-scale solves can be parallelized easily.

Using a spectral method on the small scale converts the partial differential equations (PDEs) into a system of 1^{st}-order ordinary differential equations (ODEs). The evaluation of the right-hand side (RHS) of these ODEs at a given time level depends only on data from a previous time level. Hence within each small-scale solve the evaluation of the Galerkin ODE RHSs can be parallelized at each time step. Parallelization can be further implemented within each RHS evaluation to calculate the nonlinear convolutions and the cross-terms. Figure 2 shows these different levels of parallelization possible in ATD.

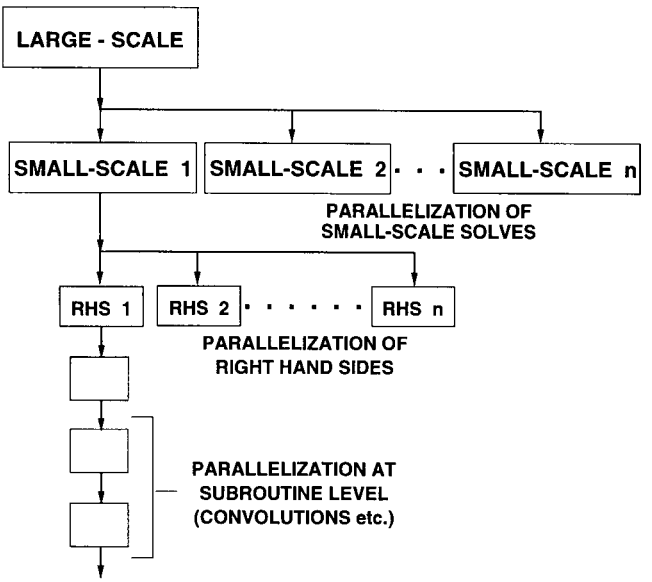


Figure 2. Levels of parallelization in ATD

3. PRESENT RESEARCH

At present the solution of only the small-scale equations is being attempted using parameterized inputs from the large-scale. This is a three-dimensional calculation in generalized coordinates. The large-scale is computed via a commercial CFD code.

The small-scale equations are solved using the Fourier-Galerkin spectral method described by Orszag [8]. In this technique the dependent variables are expressed as truncated triple Fourier series using complex exponential basis functions as

$$\boldsymbol{U}^*(\boldsymbol{\xi}, t) = \sum_{\boldsymbol{l}}^{\boldsymbol{N}} \boldsymbol{a_l}(t) \exp(i\boldsymbol{\alpha_l} \cdot \boldsymbol{\xi}), \tag{7}$$

where $\boldsymbol{U}^*$ is now the small-scale contravariant velocity vector, $\boldsymbol{l}$ is the vector of mode indices, $\boldsymbol{N}$ represents the maximum number of Fourier modes in each direction, $\boldsymbol{\xi}$ is the position vector in generalized coordinates with origin at the large-scale grid point, $\boldsymbol{a_l}(t)$ is the vector of time-dependent complex Fourier coefficients and $\boldsymbol{\alpha_l}$ is the vector of wavenumbers which are scaled based on the large-scale grid spacing.

The Fourier representations of the dependent variables are then substituted into the governing equations, and the necessary operations are performed. Taking Galerkin inner products gives the spectral projection of the equations which are now a system of 1^{st}-order ODEs in the time-dependent Fourier coefficients:

$$\frac{d\boldsymbol{a_l}}{dt} = \boldsymbol{f}(\boldsymbol{a_l}, \bar{\boldsymbol{U}}). \tag{8}$$

The function $\boldsymbol{f}$ is a notation to denote a rather complicated right-hand side vector. It should be noted that the Fourier coefficients of the small-scale solution depend explicitly on the large-scale solution $\tilde{\boldsymbol{U}}$. This system of ODE initial value problems in the Fourier coefficients is integrated forward in time using Heun's method, an explicit 2^{nd}-order Runge-Kutta scheme. Once the Fourier coefficients are known at a given time, the dependent variables can always be calculated using the Fourier series representation of equation (7).

4. RESULTS & DISCUSSION

4.1. Flow specifics & problem parameters

The particular flow problem under consideration is a three-dimensional turbulent jet. The flow is incompressible and the fluid has constant properties close to those of air: $\rho = 1.0\,\text{kg/m}^3$ and $\mu = 1 \times 10^{-6}\,\text{Ns/m}^2$. The jet diameter of 10 mm. is the reference length, and the total axial length of the domain is 760 mm. The jet velocity at the inlet is 5.0 m/s giving an inlet $Re = 5 \times 10^4$. The large-scale solution is obtained from a finite volume commercial CFD code on a rather coarse $13 \times 14 \times 81$ grid.

Figure 3 displays an instantaneous spatial slice of the large-scale solution (velocity vectors and pressure) and the location of the small-scale subdomain. The subdomain has a small radial offset, and its axial position is halfway between the inlet and outlet. This gives a non-dimensional axial location of 38, which is well beyond the potential flow core and in the region of fully-developed turbulence. Adequate resolution on the small-scale was obtained by $\boldsymbol{N} = (7, 7, 7)$ Fourier modes. The choice of the small-scale time step was dictated by the stiffness of the problem and the explicit nature of the solution method; it was set to 2×10^{-6} sec. to maintain stability.

4.2. Turbulence calculations

The main motivation for the present research is to obtain highly resolved small-scale turbulent solutions. For this purpose the code was allowed to perform time integrations for relatively long times corresponding, approximately, to the large-scale time scale. Figure 4 shows the time series of the circumferential component of small-scale velocity at a

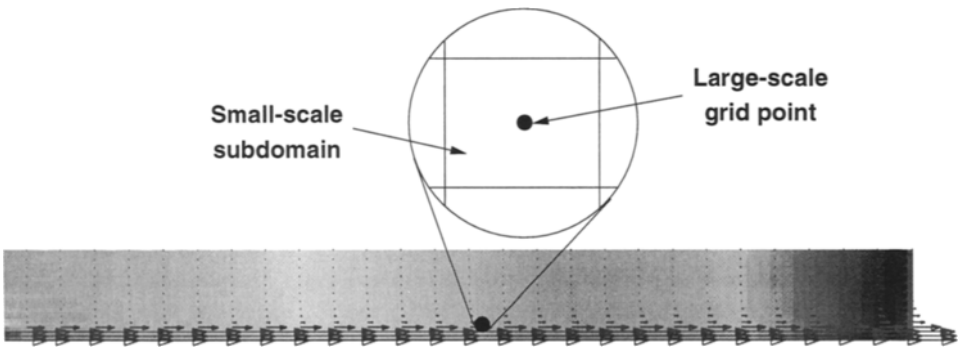


Figure 3. Large-scale flowfield and location of small-scale domain

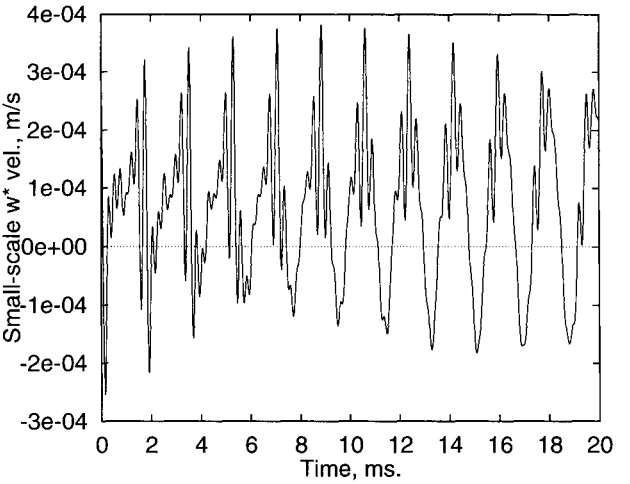


Figure 4. Time series of small-scale velocity

location in the center of the small-scale subdomain. The time series shows strong residual periodicity with a time scale of about 2 ms. This is probably to be expected in a region where the jet profile is self-similar. Moreover, these fluctuations are truly small scale having magnitudes of order 10^{-4} compared with $\mathcal{O}(1)$ for the large-scale velocity. This demonstrates the high degree of spatial and temporal resolution achievable with ATD.

4.3. Parallelization results

A broad, coarse-grained parallelization of the ODE right-hand side evaluations was implemented. The code was allowed to integrate the solution forward in time for fifty small-scale time steps. Figure 5 shows the speedups obtained on a distributed-memory Convex-HP MetaSystem consisting of 8 processors running PVM. Parallelization was also implemented using compiler directives on a 24 processor shared-memory Convex-HP Exemplar SPP-1000. The corresponding speedup results are shown in Figure 6.

The data partitioning used to divide the computational load among different processors is very effective for both machines. This is evident from the balanced CPU load on the parallel threads as shown in Figure 7. The CPU time required to solve the problem using 8 processors was about 300 sec. on the MetaSystem and about 120 sec. on the Exemplar. Thus the code runs much faster under the shared-memory paradigm, and this is despite the fact that the processors on the Exemplar currently have a slightly lower clock speed.

It can be seen that the speedups achieved with multiple processors scale essentially linearly with the number of processors. It should be recalled from Figure 2 that parallelization can be implemented at levels both above and below the present one. This implies that with the availability of true MPPs ($\mathcal{O}(10^3)$ processors, or more) the ATD algorithm can perform full three-dimensional turbulence simulations in physically realistic situations involving generalized coordinates, at Re at least as high as 10^5.

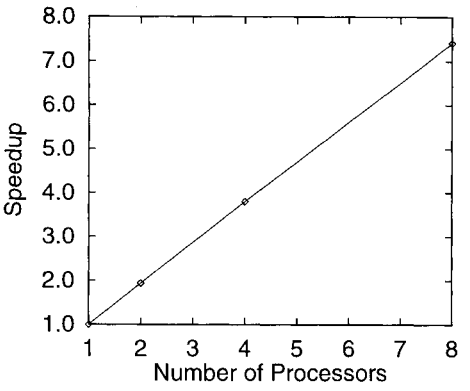


Figure 5. Parallelization speedup on Convex-HP MetaSystem

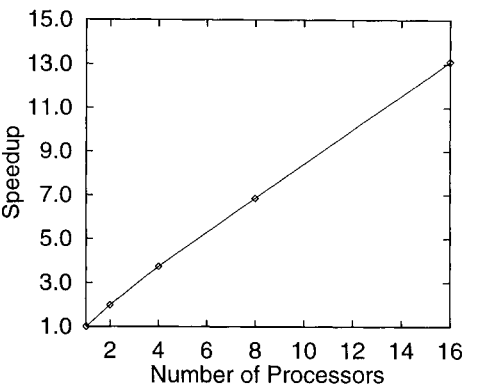


Figure 6. Parallelization speedup on Convex-HP Exemplar

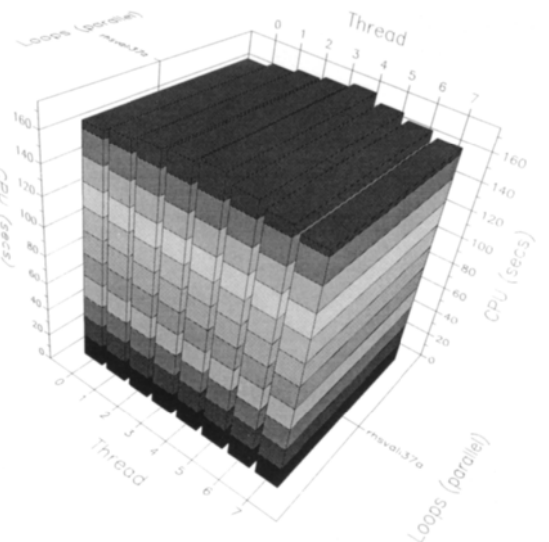


Figure 7. CPU load balance on parallel threads (Profile generated for 8 processors on Convex-HP Exemplar)

5. SUMMARY & CONCLUSIONS

This is the first implementation of the ATD algorithm in three space dimensions. The two main (theoretically expected) features of ATD are its parallelizability and its ability to produce highly resolved solutions; both are demonstrated in the present results.

Parallelization in ATD can be implemented at several different levels which can be nested. Parallelization speedups scale essentially linearly with number of processors for an intermediate (and thus more difficult to parallelize) level. This implies that high Re three-dimensional turbulence simulations are possible on MPPs.

The present code parallelizes well under both shared-memory and distributed-memory paradigms. Programming is much easier under shared memory using a few compiler directives than under distributed memory which requires learning a message passing language. The execution times are also much less under shared memory, even with somewhat slower processors. We conclude from this that, despite a rather widespread early fear of shared-memory architectures, they can perform at least as well as distributed-memory machines. We believe this makes them the obvious choice for future MPP architectures due to their ease of use.

REFERENCES

1. J. H. Ferziger, "Higher-level Simulations of Turbulent Flows," in *Computational Methods for Turbulent, Transonic and Viscous Flows*, Essers (ed.), Hemisphere Pub. Corp., Washington DC, 1983.
2. M. Germano, K. Piomelli, P. Moin and W. H. Cabot, "A Dynamic Subgrid-scale Eddy Viscosity Model," *Phys. Fluids* A3, 1760, 1991.
3. J. M. McDonough, "On the effects of modeling errors in turbulence closures for the Reynolds-averaged Navier-Stokes equations," Report CFD-03-93, Department of Mechanical Engineering, University of Kentucky, 1993.
4. J. M. McDonough and R. J. Bywater, "Large-Scale Effects on Small-Scale Chaotic Solutions to Burgers' Equation," *AIAA J.* 24, 1924, 1986.
5. J. M. McDonough and R. J. Bywater, "Turbulent Solutions from an Unaveraged, Additive Decomposition of Burgers' Equation," in *Forum on Turbulent Flows-1989*, Bower & Morris (eds.), FED Vol. 76, ASME, NY, 1989.
6. J. M. McDonough and D. Wang, "Additive turbulent decomposition: A highly parallelizable turbulence simulation technique," Report CFD-02-94, Department of Mechanical Engineering, University of Kentucky, 1994. Submitted to *Int. J. Supercomput. Appl. High Perform. Comput.*.
7. J. M. McDonough, J. C. Buell and R. J. Bywater, "An Investigation of Strange Attractor Theory and Small-Scale Turbulence," *AIAA Paper 84-1674*, 1984.
8. S. A. Orszag, "Numerical Methods for the Simulation of Turbulence," *Phys. Fluids, Supplement II*, 12, 250, 1969.
9. W. C. Reynolds, "The Potential and Limitations of Direct and Large Eddy Simulation," in *Whither Turbulence? Turbulence at the Crossroads*, Lumley (ed.), Springer-Verlag, Berlin, 313, 1990.
10. Y. Yang and J. M. McDonough, "Bifurcation Studies of the Navier-Stokes Equations Via Additive Turbulent Decomposition," in *Bifurcation Phenomena and Chaos in Thermal Convection*, Bau et al. (eds.), HTD Vol. 214, ASME, NY, 1992.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A message-passing, distributed memory parallel algorithm for direct numerical simulation of turbulence with particle tracking

P.K. Yeung and Catherine A. Moseley

School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332, U.S.A.

An efficient distributed-memory parallel pseudo-spectral algorithm for the direct numerical simulation of turbulence is presented. Timing studies illustrate the effects of various parameters on parallel performance with and without tracking fluid particles.

1. INTRODUCTION

In the last two decades, Direct Numerical Simulations (DNS, see Reynolds 1990 for a recent review)—in which the exact Navier-Stokes equations are solved numerically—have revealed much of the fundamental physics of turbulence, the most common state of fluid motion in applications. However, because of the need to resolve unsteady, three-dimensional fluctuations over a wide range of scales (which increase rapidly with Reynolds number), these simulations are extremely computationally intensive. Consequently, most known DNS results have been limited to modest Reynolds number, which has sometimes limited their significance for turbulence at the high Reynolds numbers typically encountered in practice. To achieve significantly higher Reynolds numbers requires faster calculations *and* the use of more grid points. It is clear that massively parallel computing on distributed-memory (as opposed to shared memory) architectures meets both requirements well, and therefore provides great opportunities for significant progress.

The present objective is to devise an efficient parallel implementation of a pseudo-spectral algorithm for DNS of homogeneous turbulence (Rogallo 1981). Whereas other parallel DNS algorithms are known (e.g. Pelz 1991, Chen & Shan 1992), a distinctive element in our work is a parallelized capability for tracking fluid particles and hence extracting Lagrangian statistics (Yeung & Pope 1988, 1989) from the Eulerian velocity field. The most CPU-intensive operations are Fast Fourier Transforms (FFTs) used to solve the Eulerian equations of motion, and fourth-order accurate cubic-spline interpolation which is used to obtain Lagrangian fluid particle velocities. The data partition between the parallel nodes (processors) is designed to facilitate these operations.

In the next section we describe the new parallel algorithm, and its implementation on a 512-node IBM SP2 at the Cornell Theory Center. Performance data are presented and analyzed for an FFT code, and the DNS code with and without particle tracking. Finally we briefly present representative results at 256^3 resolution showing the attainment of the well-known inertial range energy spectrum for high Reynolds number turbulence. Detailed results for multi-species mixing (Yeung & Moseley 1995) and fluid-particle dispersion (Yeung 1994) will be reported separately in the future.

2. PARALLEL ALGORITHM

We use a single program, multiple data (SPMD) approach. Because each node has a finite memory capacity (at 128 Mb for most of the nodes we use), the use of distributed-memory parallelism with a suitable data partition scheme is crucial to meeting the goal of tackling bigger problems—with up to 512^3 grid points or more. For inter-processor communication on the SP2 we use the Message Passing Library (MPL), which includes "collective" communication routines used to facilitate operations such as simultaneous data exchange and reduction across *all* nodes. For efficient parallel performance the workload on each node should, of course, be as balanced as possible. Furthermore, it is desired to minimize the time consumed by communication calls.

The data structure for the Eulerian simulation can be understood by considering how three-dimensional FFT's may be accomplished. In most cases the solution domain is a periodic cube, as shown in Fig. 1. We divide the data into a number of *slabs*, equaling the number of nodes. Fourier transforms in the y direction are taken (using the IBM-optimized ESSL library) when the data are stored as $x-y$ slabs (which consists of a number of $x-y$ planes), whereas transforms in x and z are taken when the data are in $x-z$ slabs. Transposing between $x-y$ and $x-z$ slabs (and vice-versa) requires each node to exchange a message (a "pencil" of data) with another, and is achieved by a single collective communication call that also forces automatic synchronization. Other communications calls are used in collecting global statistics such as spectra and probability density functions by combining information from all slabs.

Besides directly controlling the communication costs, the data structure also strongly affects CPU and memory requirements. Since the arrays are smaller than in the serial code, a reduction in vector strides yields significant savings in computation time. On the other hand, data residing in non-contiguous arrays must be packed into contiguous buffers before they can be sent as messages, and must be similarly unpacked afterwards. The additional memory needed for these buffers can cause memory paging penalties if the total requirement is close to that available on each node, or even render a calculation beyond the memory capacity of the system.

In the Lagrangian part of the code, to evaluate each fluid particle velocity component from an N^3 grid we form a three-dimensional tensor product spline, in the form

$$g(x,y,z) = \sum_{k=0}^{N_b-1} \sum_{j=0}^{N_b-1} \sum_{i=0}^{N_b-1} b_i(x) c_j(y) d_k(z) e_{ijk} \ , \qquad (1)$$

where $\{b_i\}$, $\{c_i\}$ and $\{d_k\}$ are sets of $N_b = N+3$ one-dimensional basis functions determined solely by the particle position coordinates (x,y,z), and $\{e_{ijk}\}$ are three-dimensional basis function coefficients which depend on the grid point values of the

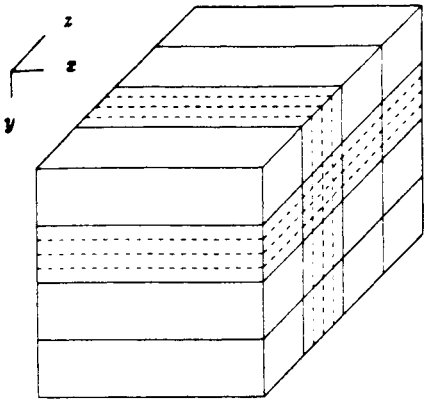


Fig. 1. Schematic showing the partitioning of solution domain into slabs for three-dimensional Fourier transforms. Both $x-y$ and $x-z$ slabs, indicated by dashed lines, consist of an equal number of data planes. The intersection between two slabs of different orientations forms a "pencil".

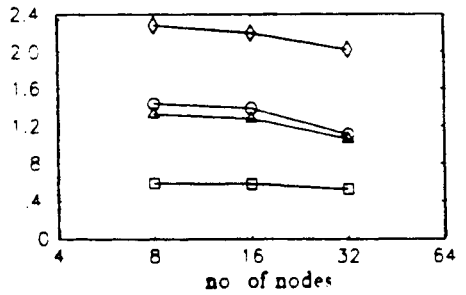


Fig 2(b) Same as Fig 2(a), but for transforms of 9 variables.

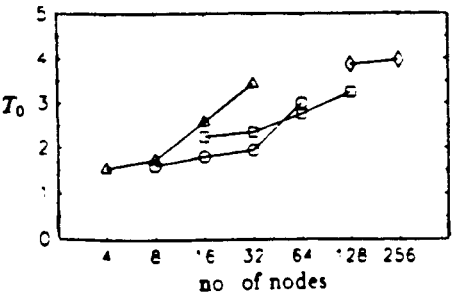


Fig. 3. Total transform time (T_0) in 10^{-6} seconds (over all nodes) per variable per grid point, vs. number of thin nodes. FFTs are taken over 64^3 ($\triangle$), 128^3 ($\bigcirc$), 256^3 ($\square$) and 512^3 ($\diamond$) grid points.

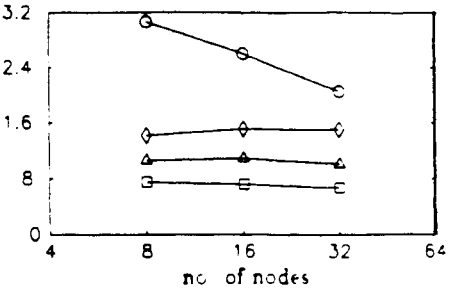


Fig. 2(a). Performance parameters vs. number of thin nodes running 3D FFT code at 128^3 and 5 variables: parallel efficiency (γ, $\triangle$) in relation to the parameters β ($\bigcirc$), $\beta/(\beta+1)$ ($\square$) and α ($\diamond$). (See. Eq. 2.)

Table 1. Comparison of performance between thin nodes and wide nodes, for 128^3 transforms with 16 nodes. Timings listed are serial time (t_s), total computation (t_c), and communication (t_m) times over all nodes, and wall time per node (t_p).

node type	thin	wide	thin	wide
N_U	5	5	9	9
t_s	22.5	8.4	51.1	28.4
t_c	10.24	7.84	33.8	29.3
t_m	7.36	5.92	13.0	9.80
t_p	1.10	0.86	2.92	2.44
$\alpha = t_s/t_c$	2.20	1.07	1.51	0.97
$\beta = t_c/t_m$	1.39	1.32	2.60	2.99
$\beta/(\beta+1)$	0.582	0.569	0.722	0.749
γ	128%	61%	109%	71%

Table 2. Comparison of total wall time (over all nodes) per Fourier transform pair per variable per grid point, between current data structure and modified scheme with reduced memory overhead.

Memory overhead	Current	Reduced
$N_U = 5$		
512^3, 128 nodes	3.86	3.15
$N_U = 9$		
128^3, 16 nodes	2.43	2.01
128^3, 32 nodes	2.56	2.19
256^3, 32 nodes	3.29	2.46
256^3, 64 nodes	2.45	2.83
512^3, 256 nodes	3.80	4.02

velocity field. Although the summation above formally ranges over N_b^3 elements, in practice for each given particle position only $4^3 = 64$ basis function values are nonzero, representing information from the four nearest grid planes in each direction. The generation of cubic spline coefficients is accomplished by solving a special system of simultaneous equations (Ahlberg *et al.* 1967), with CPU time scaling as N^3, whereas time for the summation above scales linearly with the number of particles, M.

Two different schemes for data partitioning in cubic spline interpolation have been considered. The first is for each node to carry different (sub)sets of fluid particles, but to access the same full N_b^3 array of spline coefficients. However this is very inefficient because large messages need to be passed around, and for large N this causes a severe limitation in the memory required per node. On the other hand, given that M will generally be much less than N^3 in most calculations, it is much more efficient to let each node carry all particles, but only a slab of the spline coefficients. To generate these coefficients, we first form splines in x and z directions with the Eulerian data in $x - z$ slabs, swap the partially formed results into $x - y$ slabs, and then complete the splines in y. Because N_b is not an integer multiple of the number of nodes (whereas N is), the workload is slightly uneven: some nodes are assigned one plane of spline data more than the others. Finally, for the summation in Eq. 1, since the four z-planes closest to each particle location may span two adjacent $x - y$ slabs, partially-formed interpolated results are combined from all nodes.

3. PARALLEL PERFORMANCE

All timings quoted in this paper pertain to SP2 hardware at Cornell, as of July 1995. SP2 nodes are characterized as *thin* or *wide* (Agerwala *et al.* 1995): most thin nodes have 128 Mb memory, and all have 64 Kb data caches, whereas wide nodes have at least 256 Mb memory, and 256 Kb data caches. Because of the difference in cache size, thin nodes display a great sensitivity to long vector strides, and so are generally slower. Furthermore, since the majority of nodes are thin, most production runs will be made on thin nodes, although some wide-node timings are included for comparison.

We introduce appropriate measures of parallel performance here. A conventional measure is the parallel efficiency (γ), given by the speedup relative to the serial code and divided by the number of nodes. Because of vector stride considerations as discussed, the ratio of the serial CPU time (t_s) to the total computation time (t_c) over all parallel nodes, which we denote by the "computational enhancement factor" (α), is greater than unity. Also of obvious importance is the ratio (β) of computation to communication (t_m, including pack/unpack operations) times. These parameters are related by

$$\gamma = \alpha\beta/(\beta + 1) \; . \tag{2}$$

That is, a high computational enhancement factor can lead to effective parallel efficiencies above 100%, largely as a side-benefit of the distributed-memory scheme.

3.1 FFT timings

Since FFTs and the associated communication operations are found to take up to 85% of the elapsed wall time in the Eulerian simulation, a detailed timing analysis with

a simple FFT code yields valuable insights into different factors affecting parallel performance. The FFT timings are also good indicators of the feasibility of 512^3 simulations to be performed. To mimic the DNS data structure we consider transform times for 5 variables ($N_U = 5$, corresponding to DNS with hydrodynamic field only) and 9 variables ($N_U = 9$, for DNS with three passive scalars added).

Figure 2(a) and (b) show thin node performance for a pair of FFTs at $N_U = 5$ and 9 respectively, for 128^3 grid points on 8, 16 and 32 nodes, via the quantities α, β, γ —and $\beta/(\beta + 1)$, which would be equal to γ if serial and parallel computation times were the same. Parallel efficiencies greater than 100% are achieved, mainly as a result of high computational enhancement factors as explained above. As expected, γ decreases when a relatively large number of nodes is used, mainly due to increased total communication time incurred by all nodes sending and receiving a larger number of smaller messages. The net effect of an increase in N_U on the data structure is a larger vector stride, in both serial and parallel codes (for the latter, especially if the number of nodes is small). This tends to increase the computation time substantially —more than in proportion to N_U, whereas the communication time is proportional to N_U. Consequently, for higher N_U we find lower computational enhancement factors and, despite higher computation-to-communication ratios, lower parallel efficiencies.

To compare their performance, we show detailed 16-node 128^3 timings for both thin and wide nodes in Table 1. The computational enhancement factor is seen to be especially significant for thin nodes, giving very high apparent parallel efficiencies. However, the ratio t_c/t_m varies little between the two types of nodes, suggesting that wide nodes both compute and communicate faster and by about the same factor. The parallel efficiency of wide nodes is mainly determined by this ratio, increasing with N_U.

Whereas speedup and parallel efficiency measure performance relative to a serial code, absolute performance in terms of elapsed time for a given problem size is also important, and would also provide the basis for comparisons between different algorithms and hardware platforms. In Fig. 3 we show the total transform time (sum of wall time over all nodes) per variable *per grid point*, denoted by T_0, for thin nodes as N varies from 64 to 512. Because the computation operation count scales as $N^3 \ln_2 N$, T_0 is expected to increase with N (approximately as $\ln_2 N$). For a given N, the variation of T_0 with number of nodes depends on two factors. If too few nodes are used (e.g., 512^3 with less than 128 nodes), large memory requirements per node may make the computation impossible, or degrade the performance as a result of memory paging and/or an acutely adverse impact of long vector strides. On the other hand, using more nodes causes an increase in communication time which tends to reduce the parallel efficiency. This effect accounts for the rapid rise of T_0 observed for 64^3 and 128^3 at large number of nodes. For operational purposes, for each given problem size (N), an optimum choice for the number of nodes, increasing with N, can be made when all performance measures (including T_0 and γ) are considered.

A limitation of the current parallel FFT code (which forms the basis of DNS codes in use) is a significant memory overhead (see Sec. 2). This memory overhead is relatively large because lines of data for all flow variables are stored next to each other, and is partly responsible for the large T_0 observed for 512^3 FFTs in Fig. 3. To reduce the memory requirements, we have tested a modified data structure in which different flow variables

are stored in entirely segregated memory areas on each node. Selected timings for the two data structures are compared in Table 2. The modified data structure gives shorter vector strides that are independent of N_U, but carries higher communication costs. Consequently, the new scheme is efficient when computation costs are high compared to communication (e.g, 512^3 with 128 nodes), whereas the current scheme is still favored when the memory overhead is not limiting (e.g., 512^3 with 256 nodes).

3.2 Performance of DNS codes

As already mentioned, there is a very close relationship between the performance of the FFT and DNS codes. In particular, since a pair of Fourier transforms is taken for each of two Runge-Kutta steps in the time advance scheme, the communication time per time step is just twice that measured in the FFT code. On the other hand, the computation time now includes additional contributions from other algebraic operations, such as forming products in physical space for the nonlinear terms of the equations of motion. Since the latter operations essentially have stride one, the DNS code overall displays a less marked dependency on vector stride. Consequently, the performance difference between thin and wide nodes is expected to be less than in the FFT code.

Figure 4 shows 128^3 timing information for the Eulerian parallel DNS code without scalars, corresponding to Fig. 2(a) for the FFT code. The computation enhancement factor (α) is less than for the FFT code, because of lesser sensitivity to long vector strides as discussed above. However the ratio t_c/t_m is higher, since now more computations are performed. Yet, the overall effect is a reduced (but still high) observed parallel efficiency.

Of greater interest, of course, is the performance of the DNS code at high numerical resolution, such as with 256^3 grid points. In Fig. 5 we show the observed parallel efficiency for several 128^3 and 256^3 cases. Data points linked by the solid lines indicate that wide nodes give lower efficiency (because of less computational enhancement), and that efficiency increases with the number of scalars (because of more computation relative to communication). However, because wide nodes are limited in number and serial codes on thin nodes are limited to 128 Mb memory, truly unambiguous parallel efficiencies are not available for high-resolution cases with large number of nodes. In such cases, we resort to comparing parallel times on thin nodes with serial times on wide nodes. In doing so, the performance of the code is somewhat understated. Nevertheless, using 64 thin nodes the wall time for 256^3 is only about 7.5 seconds per time step (increasing to 15 if three scalars are included), which is comparable to (serial) CPU times for 64^3 simulations on many workstations in use today. If enough wide nodes were available we estimate a wall time reduction in the range 10-30%.

To assess the performance of our parallelized particle-tracking algorithm, we consider the effect of a large number of particles (98304, the number used in Yeung 1994) on the parallel efficiency. Parallel times on thin nodes are compared with serial times on wide nodes in Fig. 6. It is seen that, relative to the Eulerian code without particles, the parallel efficiency is increased when the number of grid points and number of nodes are both small, but reduced for high-resolution simulations with large number of nodes. This suggests that the parallel Lagrangian algorithm does not scale very well with increasing number of nodes, which is largely a result of the communication time involved in combining partially interpolated results from all nodes. However, the wall time per

node is still sufficiently low to make the computations feasible.

4. NUMERICAL RESULTS

We show only very brief results here. A primary motivation for performing high-resolution simulations is to reach Reynolds numbers high enough to show inertial-range scaling behavior. Figure 7 shows the energy spectrum in wavenumber (k) space, for stationary isotropic turbulence at Taylor-scale Reynolds number (R_λ) 160, on a 256^3 grid (using 64 nodes). The spectrum is scaled by the Kolmogorov variables, such that a flat plateau signifies well-known $k^{-5/3}$ scaling at intermediate wavenumbers, and its height yields a Kolmogorov constant at about 2.0. Both the Reynolds number and the number of grid points are similar to a previous simulation by Vincent & Meneguzzi (1991) on a traditional vector supercomputer. However, with high parallel speedups such results can now be obtained in significantly reduced wall times. The value of R_λ we reached is higher than that in typical laboratory wind-tunnel experiments.

5. CLOSURE

We have implemented a distributed-memory parallel code for direct numerical simulation of turbulence with particle tracking, with parallel efficiency at more than 50% in many cases. Detailed analyses of computation and communication time have been made, especially for a Fourier transform code used to investigate the feasibility of simulations up to 512^3. Such high-resolution simulations on the IBM SP2 will greatly extend current work on multi-species scalar transport (Yeung & Pope 1993, Yeung & Moseley 1995) and fluid-particle dispersion (Yeung 1994).

We gratefully acknowledge support from the National Science Foundation, Grants CTS-9307973 and CTS-9411690, and the U.S. Environmental Protection Agency, Agreement No. R 821340-01. The second author is also supported by an NSF Graduate Fellowship Award. Supercomputing resources were provided by the Cornell Theory Center, which receives major funding from NSF and New York State. We also thank our assigned consultant at Cornell, Dr. John A. Zollweg, for his extremely valuable assistance.

REFERENCES

AGERWALA, T., MARTIN, J.L., MIRZA, J.H., SADLER, D.C., DIAS, D.M. & SNIR, M. (1995) *IBM Sys. J.* **34**, 152-184.

AHLBERG, J.H., WILSON, E.N. & WALSH, J.L. (1967) *The Theory of Splines and Their Applications*, Academic Press, New York.

CHEN, S. & SHAN, X. (1992) *Comput. Phys.* **6**, 643-646.

PELZ, R.B. (1991) *J. Comp. Phys.* **92**, 296-312.

REYNOLDS, W.C. (1990) The potential and limitations of direct and large eddy simulations. In *Whither Turbulence? Turbulence at the Crossroads*, edited by J.L. Lumley, Springer-Verlag, New York.

ROGALLO, R.S. (1981) NASA TM 81315.
VINCENT, A. & MENEGUZZI, M. (1991) *J. Fluid Mech.* **225**, 1-20.
YEUNG, P.K. (1994) *Phys. Fluids* **6**, 3416-3428.
YEUNG, P.K. & MOSELEY, C.A. (1995) *AIAA Paper* 95-0866.
YEUNG, P.K. & POPE, S.B. (1988) *J. Comp. Phys.* **79**, 373-416.
YEUNG, P.K. & POPE, S.B. (1989) *J. Fluid Mech.* **207**, 531-586.
YEUNG, P.K. & POPE, S.B. (1993) *Phys. Fluids A* **5**, 2467-2478.

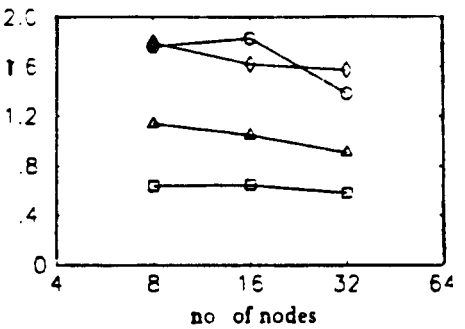


Fig 4. Same as Fig 2(a), but for parallel Eulerian DNS code with 128^3 grid points and hydrodynamic field only

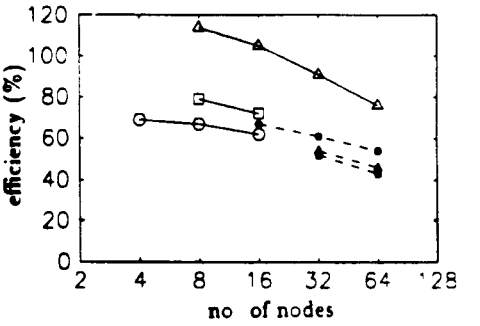


Fig. 5. Observed parallel efficiency vs. number of nodes for parallel Eulerian DNS code: 128^3, no scalars, thin nodes (△); 128^3, no scalars, wide nodes (○); 128^3, 3 scalars, wide nodes (□); 128^3, 3 scalars, thin nodes (•); 256^3, no scalars, thin nodes (□); 256^3, 3 scalars, thin nodes (▲). The closed symbols and dashed lines denote parallel thin nodes versus serial wide nodes.

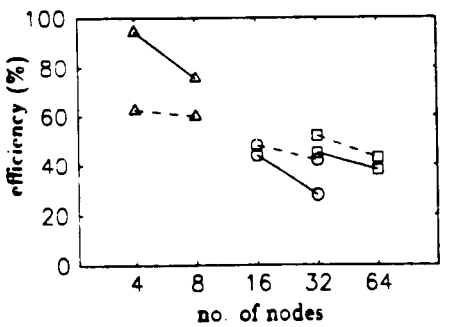


Fig 6. Observed parallel efficiency vs. number of thin nodes (compared with serial wide node) for parallel DNS code with 98,304 fluid particles: for 64^3 (△), 128^3 (○), and 256^3 (□) grid points. Dashed lines show timing data without particles, for comparison.

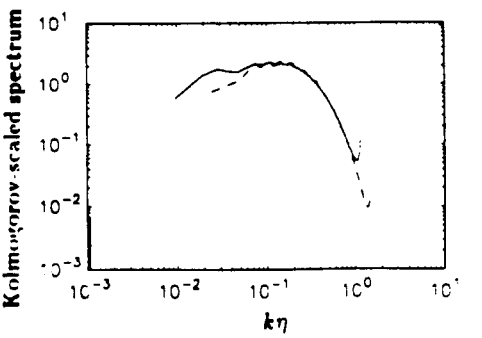


Fig. 7. Kolmogorov-scaled energy spectrum at R_λ 160 obtained from 256^3 simulation, with time averaging. Wavenumber is normalized by the Kolmogorov scale (η), and the dimensionless spectra have been multiplied by $k^{5/3}$. Dashed line shows similar data for 128^3 at R_λ 90.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Simulation Of Stratified Turbulent Channel Flows On The Intel Paragon Parallel Supercomputer

Rajat P. Garg, Joel H. Ferziger and Stephen G. Monismith[a]

[a]Environmental Fluid Mechanics Laboratory,
Department of Civil Engineering, Stanford University, Stanford, CA 94305

Implementation of a hybrid spectral finite-difference algorithm for computation of variable density turbulent flows on distributed memory message-passing computers is presented. A single-program multiple data abstraction is used in conjunction with static uni-partitioning scheme. Speedup analysis of a model problem for three different partitioning schemes, namely, uni-partition, multi-partition and transpose-partition is presented and it is shown that uni-partitioning scheme is best suited for this algorithm. Unscaled speedup efficiencies of up to 91% on doubling the number of processors and up to 60% on an eight-fold increase in the number of processors were obtained for several different grids on iPSC/860. Absolute performance up to 15-17% of theoretical peak performance was obtained on Paragon. Comparisons between the two machines are presented.

1. Introduction

The majority of flows in nature (oceanic and atmospheric flows) are turbulent and density-stratified. Stratification has a pronounced effect, making stratified turbulent flows markedly different from unstratified flows. One of the most important effects of stratification is on mixing and entrainment. These processes play an important role in the energy balance of the oceanic upper mixed layer and the energy exchange between the atmosphere and the ocean. Consequently, a better understanding of this phenomenon is essential for accurate modeling of the oceanic mixed layer. The aim of this work is to study the effects of stratification on turbulent channel flow *via* Direct Numerical Simulation (DNS) and Large Eddy Simulation (LES) based on the incompressible Navier-Stokes and scalar transport equations.

In this paper we present the parallel implementation (on iPSC/860 and Paragon) of a spectral–finite difference method for solution of Navier–Stokes equations for a channel flow. The spatial differencing requires distributed two dimensional FFT's as well as parallel stencil type operations, which require very different communication patterns. The solution algorithm also leads to different types of data dependencies at various stages and requires different (and mutually conflicting) implementation strategies. The choice of appropriate partitioning scheme is thus crucial in obtaining optimal performance for all the stages of the algorithm. A performance model is developed for a model problem which is computationally representative of the requirements of the solution algorithm for the channel flow problem. The performance model is then used to evaluate the suitability of

different partitioning schemes applied to this problem. We begin by briefly presenting the numerical method and parallel implementation in section 2 followed by partitioning analysis in section 3. Timing results are presented in section 4 and finally, some conclusions are drawn in section 5.

2. Numerical implementation

In this section we briefly describe the numerical method used to solve the equations of motion for an incompressible Boussinesq fluid, namely the Navier-Stokes and scalar transport equations for a plane channel,

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{1}$$

$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}(u_i u_j) = -\frac{\partial p}{\partial x_i} + \frac{1}{Re}\frac{\partial^2 u_i}{\partial x_j \partial x_j} - Ri_g(\rho_* - \rho_b)\delta_{i3} \tag{2}$$

$$\frac{\partial \rho_*}{\partial t} + \frac{\partial}{\partial x_j}(\rho_* u_j) = \frac{1}{RePr}\frac{\partial^2 \rho_*}{\partial x_j \partial x_j}, \tag{3}$$

where Re, Pr and Ri_g are the Reynolds, Prandtl and bulk Richardson numbers, respectively.

The spatial discretization is based on Fourier spectral methods in the horizontal directions and second order finite differences in the vertical direction. The mesh is cartesian and is non-uniform in the vertical and uniform in the horizontal directions. The time-advancement scheme is of mixed type: Crank-Nicolson for the vertical viscous terms and RK3 (Runge-Kutta) for the nonlinear and horizontal viscous terms. Aliasing errors in the pseudospectral treatment of nonlinear terms are removed using the $\frac{2}{3}$ rule. The solution technique is based on the fractional step method (see [1]). Mathematically this algorithm can be broken into three stages; the first stage is the calculation of the explicit right hand side,

$$\mathbf{r}_v = -RK3[H(\mathbf{v}^n, {\rho_*}^n)] + \frac{1}{Re}L_v(\mathbf{v}^n); \tag{4}$$

the second stage is the velocity step (solution of intermediate velocity),

$$\left(1 - \frac{\Delta t}{2Re}L_v\right)\frac{\mathbf{v}^* - \mathbf{v}^n}{\Delta t} = \mathbf{r}_v, \tag{5}$$

and the third stage is the solution of pressure Poisson equation followed by pressure correction,

$$D^T G(p\Delta t) = D^T(\mathbf{v}^*), \tag{6}$$

$$\mathbf{v}^{n+1} - \mathbf{v}^* = -G(p\Delta t). \tag{7}$$

Here G, D^T and L_v are the discrete gradient, divergence and vertical second derivative operators, respectively. H represents the nonlinear and the horizontal viscous terms. The boundary conditions have already been incorporated in the discrete matrix operators. Note, these three stages have to be repeated for each sub-step of the $RK3$ time-advancement scheme.

The scalar equation is also solved in two stages,

$$\mathbf{r}_{\rho_*} = -RK3[H_{\rho_*}(\mathbf{v}^n, {\rho_*}^n)] + \frac{1}{RePr} L_v({\rho_*}^n), \tag{8}$$

and,

$$\left(1 - \frac{\Delta t}{2RePr} L_v\right) \frac{{\rho_*}^{n+1} - {\rho_*}^n}{\Delta t} = \mathbf{r}_{\rho_*}. \tag{9}$$

For parallel implementation the programming model is based on a node-only single program multiple data (SPMD) abstraction [2]. A static 2D uni-partitioning scheme [3] is used for data decomposition *i.e.* the three-dimensional computational domain is divided into P (total number of processors) non-overlapping subdomains (with equal numbers of points), each of which resides in a separate processor. The number P is factored into $P_x \times P_z$, the numbers of processors used in the partitioning in the x (streamwise) and z (vertical) directions, respectively. The processors exchange information at domain boundaries *via* message passing. 1D data partitioning (either the vertical or the streamwise direction distributed) can be implemented simply by setting the partition size in one of the directions to unity.

Parallel 2D FFT's required in the explicit stage of the algorithm for the 2D uni-partitioning scheme are performed using the transpose method. In this method the FFT's are only performed on memory-resident data; data in the distributed direction are brought into the local memory of a processor by performing a distributed matrix transpose (we use the multiphase complete exchange algorithm; presented in [4]). The tridiagonal system of equations arising in both the second and third steps are solved using the pipelined Thomas algorithm ([3]). Unscaled speedup efficiencies up to 81% on doubling the number of processors (keeping problem size fixed) were obtained on iPSC/860 using this algorithm (see [5]).

The accuracy of the code was verified by computing the instantaneous growth rate and phase speeds of small amplitude disturbances in an unstratified flow, and comparing with the linear theory. We also performed a large-eddy simulation of turbulent channel flow with passive scalar at Reynolds number 180 (based on friction velocity and channel half-width) and Prandtl number 0.71, on a $64 \times 64 \times 65$ grid (x, y, z respectively) and 32 processors. The results (first and second order turbulence statistics) are in good agreement with the DNS results of [6] and [7]. The code is now being used for simulations of stably stratified channel flows.

3. Speedup analysis of model problem

In order to evaluate the optimal partitioning scheme for this problem, speedup analysis of a simpler model problem, namely the scalar transport equation (equation 3) was performed. For a given velocity field, this equation is solved in two stages (equations 8 and 9). The computational characteristics of these stages are identical to the first two stages (equations 4 and 5) of the fractional step algorithm. For the present spatial discretization method, the solution of pressure Poisson equation (stage 3, equation 6) is the least expensive step for the serial (and parallel) implementation, and the total computational cost is dominated by the first two stages of the fractional step algorithm. This stage

requires inverting a tridiagonal system of equations for each wavenumber pair to obtain the Fourier coefficients of pressure, which can then be used to enforce discrete incompressibility directly in the wavenumber space (equation 7). So pressure is not required to be computed in the real space except for the purposes of computing statistical quantities involving pressure (in turbulent flow simulations). Thus the performance analysis of the scalar transport equation is representative of and relevant to the choice of the partitioning scheme for the fractional step algorithm.

We analysed three different schemes, namely uni-partitioning, multi-partitioning and the transpose-partitioning (see figure 1) applied to the model problem. The distributed FFT's required in the uni & multi-partition schemes are assumed to be performed using the transpose method. The distributed tridiagonal system of equations arising in the uni-partition scheme (in the implicit stage) is assumed to be solved using pipelined Thomas algorithm. For the multi-partition scheme the Thomas algorithm can be applied in a straight-forward fashion without any load-imbalance. In the transpose-partition both the stages are performed in-place and a global data transpose is used in-between the stages to bring all the data required in each stage, in the local memory of the processor.

The speedups and the speedup efficiencies for the different schemes on a hypercube architecture[1] can be expressed as

$$S_p = \frac{1}{\frac{1}{P} + O_s} \tag{10}$$

$$S_\eta = \frac{S_p}{P}, \tag{11}$$

where $O_s = \frac{Num_s}{Den}$ represents the parallelization overhead for each data-mapping scheme. The Num_s and Den are given below:

$$\begin{aligned} Num_{uni} &= 8(P_x - 1)\lceil\frac{N_z}{P_z}\rceil\Big[\lambda + \frac{N_xN_y}{{P_x}^2}b\beta + \delta\frac{(\log_2 P_x)P_x}{2(P_x-1)}\Big] \\ &\quad + \Big(\frac{N_xN_y}{P_x l_n} + P_z + 8\Big)\lambda + \Big(\frac{12N_xN_y}{P_x} + 4P_z l_n\Big)b\beta + 8N_z l_n t_c \end{aligned} \tag{12}$$

$$\begin{aligned} Num_{mul} &= 8(P-1)N_z\Big[\lambda + \frac{N_xN_y}{P^2}b\beta + \delta\frac{(\log_2 P)P}{2(P-1)}\Big] \\ &\quad + 6(P-1)\lambda + 12(P-1)\frac{N_xN_y}{P}b\beta \end{aligned} \tag{13}$$

$$\begin{aligned} Num_{trp} &= 2(P-1)\Big[\lambda + \frac{N_xN_yN_z}{P^2}b\beta + \delta\frac{(\log_2 P)P}{2(P-1)}\Big] \\ &\quad + 8(\lambda + N_xN_yb\beta) \end{aligned} \tag{14}$$

$$Den = \frac{N_xN_yN_z}{P}\Big[40 + 6(\log_2 N_x + \log_2 N_y)\Big]t_c. \tag{15}$$

Here N_x, N_y and N_z are the number of grid-points in the x, y and z directions, respectively. l_n represents the number of line-solves per message in the pipelined Thomas

[1] A hypercube architecture is assumed to perform the complexity analysis of the global exchange algorithm for distributed matrix transpose. Analysis for other network topologies, such as the mesh network can be done in a similar fashion.

algorithm (see [3] and [5]). The different machine parameters are t_c – the computation cost per floating point operation, λ – the message-passing latency, b – number of bytes per word of data, β – the message transfer time in *sec/bytes* and δ – distance impact in *sec/dimension* on the hypercube (see [4]). $P = P_x P_z$ is the total number of processors. The subscripts *uni*, *mul* and *trp* denote the uni, multi and transpose partitionings, respectively.

The S_η predictions for the model problem using the machine parameter values for iPSC/860 are shown in figure 2. The smaller grid is the typical grid used in large-eddy simulations and the larger grid is typical of the ones used in direct-simulations. The results clearly indicate that multi-partitioning leads to very low efficiencies compared to uni and transpose schemes. The reason for this is that while multi-partition eliminates the data-dependency in the solution of tridiagonal system of equations, it significantly increases the amount and the stages of communication required to perform the FFT's. Furthermore, it serializes the inherent parallelism in the FFT's that is exposed by the 1D vertical uni-partition or transpose schemes (groups of horizontal planes reside in each processor and in-place FFT's can be performed in each such processor independently).

For the two grids considered in figure 2, the memory requirements of the Navier-Stokes code are such that a minimum of 4 processors are required for the smaller grid and a minimum of 64 processors are required for the larger grid (for 16 Mbytes/node computers). So we compare the two schemes for greater than this minimum number in order for the predictions to be applicable to the channel flow code. For the smaller grid, the uni-partitioning shows a higher efficiency because a 1D-z partition is used and the FFT's are performed locally with communication and data-dependency delay occurring only in the vertical operations. The transpose partition on the other hand requires global communication in the transpose phase.

In the case of the larger grid a 2D uni-partitioning[2] has to be used in order to improve the efficiency of the tridiagonal solver. Thus distributed FFT's are required in the stage 1, leading to increased communication overhead. However, in the uni-partition scheme the matrix transpose (for FFT's) is performed only across P_x processors, whereas in the transpose scheme it is performed across P processors. Thus for the uni-partitioning scheme the cost of FFT's is fixed with increasing P (P_x fixed) and the nearly linear decrease in S_η is occurring due to increased overhead in tridiagonal solver.

So for the transpose partition the total number of stages ($P - 1$ on a Hypercube) as well as the data to be communicated is higher in the global exchange algorithm than the uni-partition scheme, resulting in a poorer scalability for P greater than 64 (figure 2. Furthermore, on a 2D mesh (*i.e.* on Paragon) the global exchange would suffer greater contention & require more stages than on a Hypercube. Thus based on these considerations, over the range of problem and processor sizes that are of interest to us for stratified turbulent channel flow simulations, the uni-partitioning scheme is superior than the transpose partitioning scheme, and was used in the parallel implementation.

[2] $P_x = 4$ gave the best results.

4. Results

In this section the results of timing measurements of the parallel implementation are presented. These measurements were made on a 32 node iPSC/860 Hypercube computer at Stanford and the 400 node Paragon XP/S at San Diego Supercomputer Center (SDSC). The results were obtained by advancing the solution for 50 time-steps (for each run) to obtain averaged execution times. The velocity field was initialized as low-amplitude (5% of mean) divergence-free random perturbations superposed on laminar flow profile.

The measured relative speedup efficiency S_η *vs.* number of processors P is shown in figure 3 (on iPSC/860[3]) for different grids and for 1D-z partitioning. S_η is defined as $\frac{T_{min}}{T_P} \times \frac{P_{min}}{P}$ where P_{min} is the minimum number of nodes on which a given problem size fits, T_{min} is the CPU time for P_{min} processors and T_P is the measured time for P processors. An estimate of single processor time can be made using the procedure outlined in [8] but it requires accurate measurements of the communication, computation, and idle time of each processor. This is a complicated and error-prone task for the algorithm considered here. The relative speedup efficiency is easier to compute and gives an accurate indication of the scalability of the algorithm. As expected, the efficiency increases with grid size and eventually reaches a plateau on which the CPU time increases in proportion to the number of grid points. Unscaled speedup efficiency of up to 91% on doubling the number of processors and up to 60% on an eight-fold increase in the number of processors, was obtained for the 1D vertical decomposition.

The absolute performance of the code was evaluated by computing the speed of execution based on the operation count of the algorithm[4] and the measured CPU times. The percent of peak theoretical performance (for 64–bit arithmetic) is shown as a function of number of processors for several different grids in figure 4; absolute performances up to 15–17% of theoretical maximum were obtained. Comparisons between Paragon and iPSC/860 for several grids showed that Paragon (for OSF 1.2) is on an average 30–50% faster than iPSC/860 with the higher end improvements observed mostly for problems where 2D partitioning was used. This is an expected result because the communication requirement for 2D partitioning is significantly higher (due to distributed FFT's) than the $1D-z$ partitioning and Paragon primarily is superior than iPSC/860 in communication performance.

5. Conclusions

A scalable parallel algorithm for spectral–finite difference simulation of turbulent stratified flows was implemented and its performance was evaluated. The performance was found to depend strongly on the problem size and the type of data decomposition. Three types of partitioning schemes were investigated for a model problem and the analysis showed that multi-partition schemes are ill-suited for this algorithm. Uni-partitioning schemes (1D and 2D) were found to be optimal for this problem and unscaled speedup efficiencies of up to 91% were obtained for several different grids. On the Paragon single

[3]Similar results were obtained on Paragon under OSF 1.2

[4]a conservative estimate of $786+84(2log_2N_x+4log_2N_y)$ floating point operations per grid-point was used for the 3 stage RK3-CN step.

node execution speeds between 6–12 Mflops (out of 75 Mflops peak) were obtained for the overall algorithm. The comparison between iPSC/860 and Paragon showed 30–50% faster timings on Paragon.

Acknowledgments

The authors extend thanks to Prof. R.W. Dutton for providing access to the iPSC/860 Hypercube operated by his group, to Dan Yergeau for his help with getting us started on the iPSC/860, to Dr. J.P. Singh of Stanford Computer Science department and Mr. Ken Steube of SDSC for many helpful discussions on parallel implementation. The support of SDSC for providing computer time on Paragon is also greatfully acknowledged. Financial support was provided by ONR under grant No. N00014-92-J-1611.

REFERENCES

1. J. Kim & P. Moin, Jour. of Comp. Physics, **59** (1985), p. 308.
2. F. Darema, D.A. George, V.A. Norton & G.F. Pfister, Parallel Computing, Vol. **7**, (1988) p. 11.
3. N.H. Naik, V.K. Naik & M. Nicoules, Intl. Jour. of High Speed Computing. Vol. **5**, No. 1 (1993), p. 1.
4. S.H. Bokhari, NASA Contractor Report No. 187499 (1991), ICASE, NASA Langley Research Center.
5. Garg, R.P., Ferziger, J.H. & Monismith, S.G., In Proceedings of VII SIAM Conf. on Parallel Processing for Scientific Computing, San Francisco, CA, February 1995.
6. Kim, J., Moin, P. & Moser, R.D., Jour. of Fluid. Mech. vol. **177**, p. 133.
7. J. Kim & P. Moin, In Turbulent shear flows 6, ed. by J.C. André, J. Cousteix, F. Durst, B.E. Launder, F.W. Schmidt & J.H. Whitelaw, Springer Verlag (1989), p. 86.
8. Gustafson, J.L., Montry, G.R. & Benner, R.E., SIAM Jour. of Sci. Stat. Computing, Vol. **9**, (1988), p. 609.

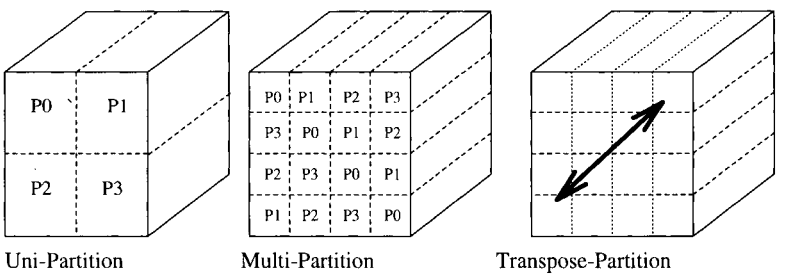


Figure 1. Uni, multi and transpose partition schemes.

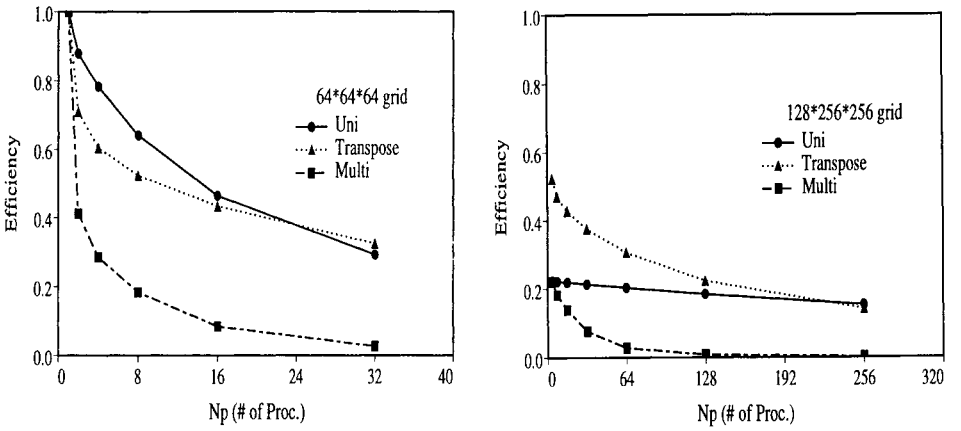


Figure 2. S_η predictions for the model problem on iPSC/860 for coarse and fine grids

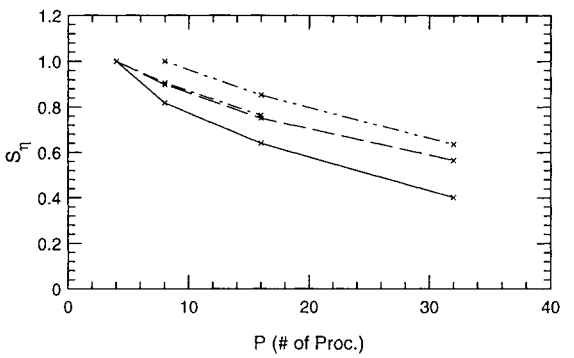


Figure 3. Speedup efficiency S_η *vs.* number of processors for iPSC/860 Hypercube, 1D-z partitioning; —— : $64 \times 64 \times 64$ grid; - - - : $64 \times 32 \times 128$ grid; —·— : $32 \times 32 \times 144$ grid; —··— : $64 \times 64 \times 128$ grid.

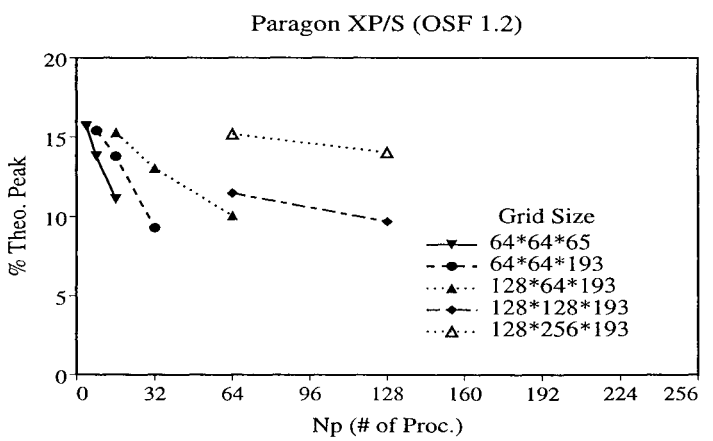


Figure 4. Theoretical peak performance (for 64-bit arithmetic) on Paragon XP/S for different grids.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel implementation of an adaptive scheme for 3D unstructured grids on a shared-memory multiprocessor

R. Biswas[a*] and L. Dagum[b]

[a]Research Institute for Advanced Computer Science,
Mail Stop T27A-1, NASA Ames Research Center, Moffett Field, CA 94035, U.S.A.

[b]Supercomputer Applications, Silicon Graphics Inc.,
Mail Stop 580, 2011 N Shoreline Blvd., Mountain View, CA 94043, U.S.A.

Dynamic mesh adaption on unstructured grids is a powerful tool for computing unsteady flows that require local grid modifications in order to efficiently resolve solution features. For such flows, the coarsening/refinement step must be completed every few time steps, so its efficiency must be comparable to that of the flow solver. For this work, we consider an edge-based adaption scheme that has shown good single processor performance on a Cray Y-MP and C90. We report on our experience porting this code to an SGI Power Challenge system and parallelizing it for shared-memory symmetric multiprocessor architectures.

1. INTRODUCTION

Unstructured grids for solving CFD problems have two major advantages over structured grids. First, unstructured meshes enable efficient grid generation around highly complex geometries. Second, appropriate unstructured-grid data structures facilitate rapid insertion and deletion of mesh points to allow the computational mesh to locally adapt to the flowfield solution.

Two types of solution-adaptive grid strategies are commonly used with unstructured-grid methods. Grid regeneration schemes generate a new grid with a higher or lower concentration of points in regions that are targeted by some error indicator. A major disadvantage of such schemes is that they are computationally expensive. This is a serious drawback for unsteady problems where the mesh must be frequently adapted. However, resulting grids are usually well-formed with smooth transitions between regions of coarse and fine mesh spacing.

Local mesh adaption, on the other hand, involves adding points to the existing grid in regions where the error indicator is high, and removing points from regions where the indicator is low. The advantage of such strategies is that relatively few mesh points need to be added or deleted at each refinement/coarsening step for unsteady problems. However, complicated logic and data structures are required to keep track of the points that are added and removed.

*Work supported by NASA under Contract Number NAS 2-13721 with the Universities Space Research Association.

For problems that evolve with time, local mesh adaption procedures have proved to be robust, reliable, and efficient. By redistributing the available mesh points to capture flowfield phenomena of interest, such procedures make standard computational methods more cost effective. Highly localized regions of mesh refinement are required in order to accurately capture shock waves, contact discontinuities, and shear layers. This provides scientists the opportunity to obtain solutions on adapted meshes that are comparable to those obtained on globally-refined grids but at a much lower cost.

Advances in adaptive software and methodology notwithstanding, parallel computational strategies will be an essential ingredient in solving complex real-life problems. However, the success of parallel computing relies on the efficient implementation of such adaptive procedures on commercially-available multiprocessor machines. Parallel performance not only depends on the design strategies, but also on the choice of efficient data structures which must be amenable to simple manipulation without significant memory contention (for shared-memory architectures) or communication overhead (for message-passing architectures).

In this work, we consider the dynamic mesh adaption procedure of Biswas and Strawn [1] which has shown good sequential performance. A description of this scheme is given in Section 2. The Euler flow solver used in the calculations is briefly described in Section 3. At this juncture, we have successfully ported the code to an SGI Power Challenge shared-memory multiprocessor with encouraging results. The algorithmic and data structure modifications that were required for the parallelization are described in Section 4. Finally, computational and parallel performance results are presented in Section 5.

2. MESH ADAPTION PROCEDURE

We give a brief description of the basic tetrahedral mesh adaption scheme [1] that is used in this work for the sake of completeness and to highlight the modifications that were made for the shared-memory implementation. The code, called 3D_TAG, has its data structures based on edges that connect the vertices of a tetrahedral mesh. This means that each tetrahedral element is defined by its six edges rather than by its four vertices. These edge-based data structures make the mesh adaption procedure capable of performing anisotropic refinement and coarsening. A successful data structure must contain just the right amount of information in order to rapidly reconstruct the mesh connectivity when vertices are added or deleted but also have a reasonable memory requirement.

At each mesh adaption step, individual edges are marked for coarsening, refinement, or no change. Only three subdivision types are allowed for each tetrahedral element and these are shown in Fig. 1. The 1:8 isotropic subdivision is implemented by adding a new vertex at the mid-point of each of the six edges. The 1:4 and 1:2 subdivisions can result either because the edges of a parent tetrahedron are targeted anisotropically or because they are required to form a valid connectivity for the new mesh. When an edge is bisected, the solution vector is linearly interpolated at the mid-point from the two points that constitute the original edge.

Mesh refinement is performed by first setting a bit flag to one for each edge that is targeted for subdivision. The edge markings for each element are then combined to form a binary pattern as shown in Fig. 2 where the edges marked with an R are the ones to be

bisected. Once this edge-marking is completed, each element is independently subdivided based on its binary pattern. Special data structures are used in order to ensure that this process is computationally efficient.

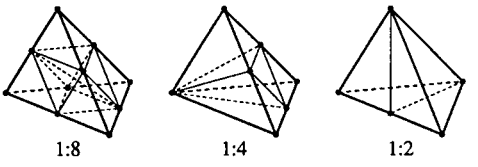


Figure 1. Three types of subdivision are permitted for a tetrahedral element.

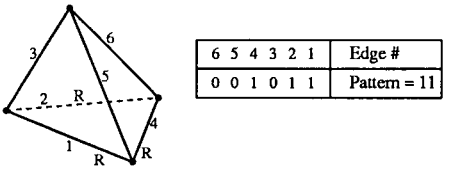


Figure 2. Sample edge-marking pattern for element subdivision.

Mesh coarsening also uses the edge-marking patterns. If a child element has any edge marked for coarsening, this element and its siblings are removed and their parent element is reinstated. The parent edges and elements are retained at each refinement step so they do not have to be reconstructed. Reinstated parent elements have their edge-marking patterns adjusted to reflect that some edges have been coarsened. The mesh refinement procedure is then invoked to generate a valid mesh.

A significant feature in 3D_TAG is the concept of "sublists." A data structure is maintained where each vertex has a sublist of all the edges that are incident upon it. Also, each edge has a sublist of all the elements that share it. These sublists eliminate extensive searches and are crucial to the efficiency of the overall adaption scheme.

Figure 3 is a schematic of the data structures used in the sequential code. For simplicity, we only show the data structures corresponding to the edges of the mesh; however, similar pertinent information is also maintained for the vertices, elements, and boundary faces. For each edge we store its two end vertices, some solver-specific geometry information, a pointer to the parent edge (if any), pointers to the two children edges (if any), color, and a pointer to the first element in the sublist of elements that share this edge. Note that `tedge` marks the first free location in all the edge arrays. The element list, however, uses two pointers, `c_edg_elm` and `t_edg_elm`. These mark the first "hole" and the start of completely free space, respectively. Holes in the element list result from coarsening; if no coarsening has taken place, `c_edg_elm` and `t_edg_elm` point to the same location. As elements are created by refinement, they are first added to the holes in the element list until those are completely filled.

An important component of any mesh adaption procedure is the choice of an error indicator. Since we are interested in computing acoustic pressure signals, we have chosen pressure differences across edges of the mesh to indicate flowfield regions that require mesh adaption. However, this error indicator does not adequately target the far-field acoustic wave for refinement because the strength of a noise signal attenuates beyond the rotor blade tip. To ensure that the relative error in the acoustic signal is evenly distributed everywhere, this error indicator must be more heavily weighted away from the rotor blade. A more detailed description of the manner in which this is accomplished is given in [2].

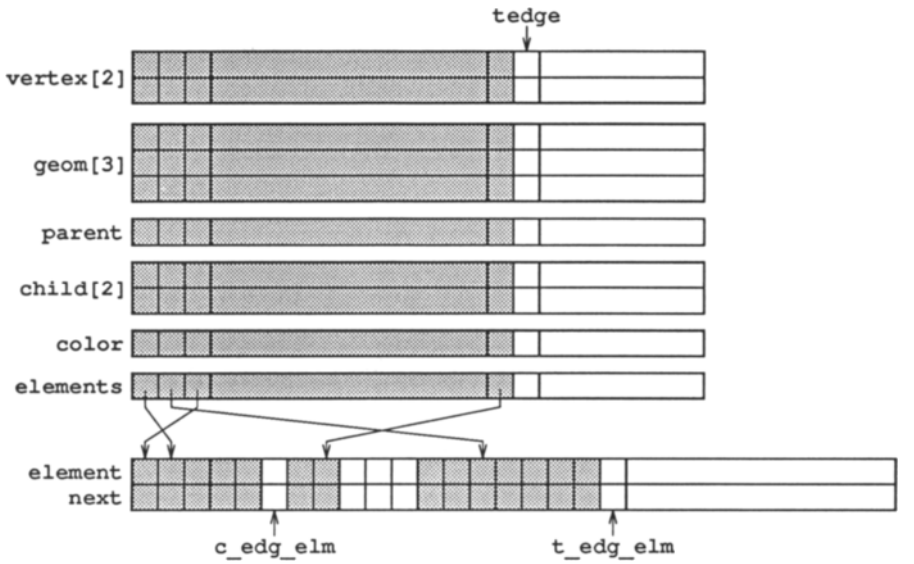


Figure 3. Edge data structures for the sequential code.

3. EULER FLOW SOLVER

The Euler flow solver, developed by Barth [3], is a finite-volume upwind code that solves for the unknowns at the vertices of the mesh and satisfies the integral conservation laws on nonoverlapping polyhedral control volumes surrounding these vertices. Improved accuracy is achieved by using a piecewise linear reconstruction of the solution in each control volume. The solution is advanced in time using conventional explicit procedures.

In the rotary-wing version [4], the equations have been rewritten in an inertial frame so that the rotor blade and grid move through stationary air at the specified rotational and translational speeds. Fluxes across each control volume were computed using the relative velocities between the moving grid and the stationary far field. For a rotor in hover, the grid encompasses an appropriate fraction of the rotor azimuth. Periodicity is enforced by forming control volumes that include information from opposite sides of the grid domain.

The code uses an edge-based data structure that makes it particularly compatible with the mesh adaption procedure. Furthermore, since the number of edges in a mesh is significantly smaller than the number of faces, cell-vertex edge schemes are inherently more efficient than cell-centered element methods [3]. Finally, an edge-based data structure does not limit the user to a particular type of volume element. Even though tetrahedral elements are used in this paper, any arbitrary combination of polyhedra can be used [5].

4. SHARED-MEMORY IMPLEMENTATION

The SGI Power Challenge XL contains a maximum of 18 64-bit 90 MHz R8000 superscalar processors, each containing 4MB of data streaming cache, and provide up to 6.48

Gflops of peak parallel performance. The shared physical memory is expandable from 64MB to 16GB with 1-, 2-, 4-, or 8-way interleaving. The memory bus can sustain a peak bandwidth of 1.2GB/sec. Details of the architecture can be found in [6].

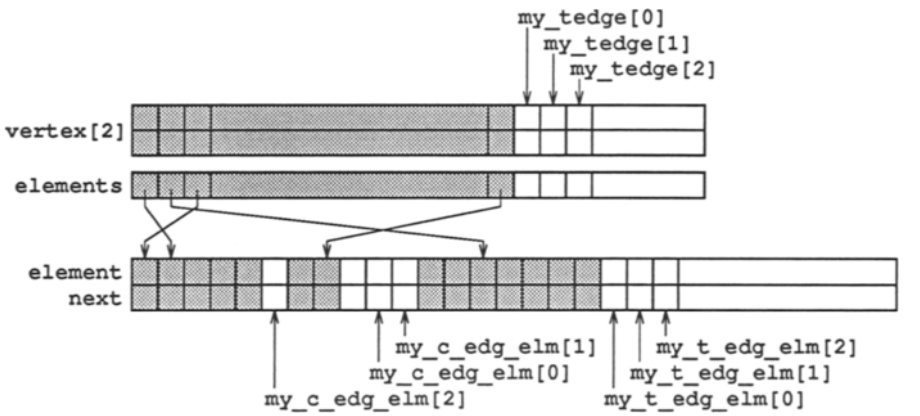


Figure 4. Edge data structures modified for the parallel code.

Figure 4 is a schematic of the modified data structures used for the parallel code. For simplicity, only the vertex and the element sublist arrays from Fig. 3 are shown. Note that `tedge` is now an array, rather than a scalar variable. For the parallel implementation, each processor `i` has an independent pointer `my_tedge[i]` to its first free location in the edge arrays. Similarly, `c_edg_elm` and `t_edg_elm` are also converted to arrays `my_c_edg_elm[i]` and `my_t_edg_elm[i]`, respectively, with an independent pointer for each processor `i`. This modification allows the processors to independently add edges and elements to the grid. These local variables are initialized as follows:

$$\texttt{my\_tedge[i]} = \texttt{tedge} + \texttt{i}$$
$$\texttt{my\_t\_edg\_elm[i]} = \texttt{t\_edg\_elm} + \texttt{i}$$
$$\texttt{my\_c\_edg\_elm[i]} = \texttt{my\_t\_edg\_elm[i]} \ \% \ \texttt{nprocs}$$

where `nprocs` is the total number of processors. Each processor increments its local variables by `nprocs` after every insertion. The `my_c_edg_elm[i]` variables are initialized the way they are in order to guarantee that they match the corresponding `my_t_edg_elm[i]` variables when all the holes are completely filled.

The main kernel of the refinement and coarsening procedures consists of looping over the list of tetrahedral elements. In a parallel implementation, this work is split among all the processors. However, it is necessary to guarantee that multiple processors will not modify the same location in the data structures. This is accomplished by coloring all the elements such that elements of the same color do not share a vertex (and consequently neither an edge). We could also have used a partitioning of the grid; however, it was simpler to enforce independence through element coloring. The advantage of a shared-memory paradigm is that it affords this flexibility in the implementation.

Coloring also has the nice additional feature of requiring only local operations on the grid. In other words, once all the elements of the initial mesh have been colored, it is necessary to color only those elements that are created from a refinement or a coarsening step, thereby greatly reducing the operation count. In the current implementation, the element coloring has not been given much attention, the focus being on obtaining parallel coarsening and refinement steps. Nonetheless, it is worthwhile briefly describing what was actually implemented.

For refinement, an element is colored by visiting each of its four vertices and then looking at all the neighbor elements of every edge incident on the vertex. The element is then assigned a color that is not used by any neighbor element. A color count array is maintained to ensure that the colors are balanced across elements. For coarsening, the procedure is slightly different in that we must also consider the parent edges sharing the vertex. This is necessary because coarsened elements are created from the parent edges and thus have a different neighbor list than is obtained through the children edges.

Once all the elements are properly colored, the parallel code sequentially loops through elements of the same color in parallel, carrying out parallel coarsening or refinement steps using independent pointers to the available openings in the data structures. The loops are dynamically scheduled with an arbitrary "chunk" size of 1000, such that each processor executes independent chunks of 1000 loop iterations. This is extremely useful for load balancing. The work associated with a single loop iteration depends on whether an element is to be coarsened/refined or not. A simple uniform splitting of the loop count across the processors typically results in a poor load balance. This is because the elements that need to be adapted tend to be concentrated in localized regions of the grid, so that some processors will have many more elements to process than others. By using a dynamic scheduling, we can get a much more balanced allocation of work to the processors.

5. RESULTS

The 3D_TAG code has been combined with the Euler flow solver to simulate the acoustics experiment of Purcell [7] where a 1/7th scale model of a UH-1H rotor blade was tested over a range of hover-tip Mach numbers, M_{tip}, from 0.85 to 0.95. These rotor speeds produce strong aerodynamic shocks at the blade tips and an annoying pattern of high-speed impulsive noise. Numerical results as well as a detailed report of the simulation are given in [2]. This paper reports only on the performance of the parallel version of 3D_TAG.

Timings for the parallel code were obtained from running only the first refinement and the first coarsening steps reported in [2]. Table 1 presents the progression of grid sizes through these two adaption steps. The code was compiled and executed with the latest released IRIX 6.1 system software on 1, 2, 4, and 8 90 MHz R8000 CPU's. The timings presented are wall clock times averaged over 10 runs on a lightly-loaded system. The original sequential code was also run on a Cray C90 and a SGI Power Challenge for comparison purposes.

Table 2 presents the timings for the various portions of the refinement and coarsening steps. The first set of results is for the first refinement that consists of marking the edges that need to be refined and then actually refining the grid. Both these times are shown separately. The second set of results is for the coarsening step. Coarsening consists of

Table 1
Progression of grid sizes through refinement and coarsening

	Vertices	Elements	Edges
Initial mesh	13,967	60,968	78,343
Mesh after refinement	35,231	179,424	220,160
Mesh after coarsening	28,987	144,719	178,645

marking edges for coarsening, cleaning up all the data structures by removing those edges and their associated vertices and tetrahedral elements, and finally invoking the refinement routine to generate a valid mesh from the vertices left after the coarsening.

Table 2
Wall clock times in seconds for the refinement and coarsening steps

	C90	SGI-seq.	SGI-1	SGI-2	SGI-4	SGI-8
`mark_to_refine()`	2.96	1.83	1.72	1.06	0.72	0.65
`refine()`	9.11	5.04	6.90	5.04	3.66	3.08
`mark_to_coarsen()`	2.99	2.02	2.08	1.57	1.28	1.20
`cleanup_arrays()`	4.29	2.30	3.35	2.21	1.45	1.05
`refine()`	6.74	3.51	6.80	4.35	2.81	2.20

On the original sequential code, we observe that the Power Challenge performs better than the C90 both for refinement and for coarsening. This is not unexpected. By nature, mesh adaption does not readily vectorize or software pipeline; therefore, both systems essentially demonstrate their scalar performance. Furthermore, most of the work is in accessing and modifying integer arrays. Because the C90 employs 64-bit integers, the amount of data motion required is about double that of the Power Challenge which uses 32-bit integers. This result serves to highlight the superior scalar performance of the Power Challenge.

The SGI parallel results, though still somewhat preliminary, are very promising. For pure refinement, both `mark_to_refine()` and `refine()` show linear speedup up to 4 processors. There is some drop-off for 8 processors, which is to be expected because of critical sections present in the code. The slow down for `refine()` in going from sequential to 1-CPU parallel execution is due to repeatedly looping over the elements for each color. A total of 43 element colors are required for the initial grid. This overhead could be avoided by sorting the elements by color; however, this has not yet been implemented.

We observe similar behavior for the coarsening step. Speedup is fairly linear up to 4 processors after which there is some drop-off. This can again be attributed to the critical sections in the code, some of which may be eventually eliminated. Because 76 element colors are required for the coarsening stage, the penalty in going from sequential to parallel execution is greater than that observed in the refinement case.

The time to color elements has not been included in Table 2 above but is a significant overhead for parallel execution and will need to be addressed. Using the highly inefficient and redundant coloring algorithm described in the previous section, coloring elements for

refinement required 9.84 seconds, and for coarsening required 7.65 seconds. These times would improve simply by saving the element coloring between executions so that only new elements have to be colored. Some simple modifications of the coloring algorithm itself would also yield improved performance.

6. SUMMARY

Fast and efficient dynamic mesh adaption is an important feature of unstructured grids that make them especially attractive for unsteady flows. For such flows, the coarsening/refinement step must be completed every few time steps, so its efficiency must be comparable to that of the flow solver. For this work, the edge-based adaption scheme of Biswas and Strawn [1] is parallelized for shared-memory architectures.

Only minor modifications to the data structures are required for a shared-memory parallel implementation. However, some scheme for ensuring independence of elements is also necessary for parallel execution. For this work, we implemented a simple element coloring scheme. A partitioning of the grid could also have been used; however, coloring was much simpler to implement. A nice feature of the shared-memory paradigm is that either scheme can be used, whereas on a distributed-memory system only grid partitioning will work.

Results are presented that compare the original sequential code on both a Cray C90 and a SGI Power Challenge and the parallel version running on a 1-,2-,4-, and 8-CPU SGI Power Challenge. For the sequential code, the Power Challenge performs about 1.8 times better than the C90. This is due in part to the Power Challenge using 32-bit integers compared to the C90 using 64-bit integers and also to the Power Challenge exhibiting better scalar performance than the C90 (given that the code neither vectorizes or software pipelines very well).

The parallel performance is promising although the observed speedups are still fairly modest. Speedups of 2.0X and 2.2X were observed on 4 processors. These tailed off to 2.3X and 2.7X on 8 processors. The parallel code still includes several critical sections that need to be removed for better performance. Also, there is a significant penalty when executing the parallel code on a single processor because the refinement and coarsening loops are executed once for every element color. Finally, these speedups do not include the time to color elements. Element coloring will be addressed in subsequent work.

REFERENCES

1. R. Biswas and R.C. Strawn, Appl. Numer. Math. 13 (1994) 437.
2. R.C. Strawn, M. Garceau, and R. Biswas, AIAA 15th Aeroacoustics Conf. (1993) Paper 93-4359.
3. T.J. Barth, AIAA 10th Comp. Fluid Dynamics Conf. (1991) Paper 91-1548.
4. R.C. Strawn and T.J. Barth, J. AHS 38 (1993) 61.
5. R. Biswas and R.C. Strawn, 3rd US Natl. Cong. Comp. Mech. (1995) 234.
6. Power Challenge Technical Report, Silicon Graphics Inc., 1994.
7. T.W. Purcell, 14th European Rotorcraft Forum (1988) Paper 2.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A Parallel Adaptive Navier-Stokes Method and Partitioner for Hybrid Prismatic/Tetrahedral Grids

T. Minyard* and Y. Kallinderis†

Dept. of Aerospace Engineering and Engineering Mechanics
The University of Texas at Austin, Austin, TX 78712

A parallel finite-volume method for solution of the Navier-Stokes equations with adaptive hybrid prismatic / tetrahedral grids is presented and evaluated in terms of parallel performance. The solver is a central type differencing scheme with Lax-Wendroff marching in time. The grid adapter combines directional with isotropic local refinement of the prisms and tetrahedra. The hybrid solver, as well as the grid adapter, are implemented on the Intel Paragon MIMD architecture. Reduction in execution time with increasing number of processors is nearly linear. Partitioning of the mesh is accomplished through recursive division of the octree corresponding to the hybrid unstructured mesh. Subdividing the octree generates partitions that have a lower percentage of elements on the boundary than if standard recursive bisection were used for the hybrid grid. Results for partitioning, solving on, and adapting a hybrid grid around an aircraft configuration are presented and discussed.

1. Introduction

The development of unstructured grid methods has aided in the simulation of flow fields around complex geometries. These methods have mostly been limited to tetrahedral grids. Tetrahedra provide flexibility in 3-D grid generation since they can cover complicated topologies easier than hexahedral meshes [1]. However, employment of tetrahedral cells for boundary layers is quite expensive. In these regions strong solution gradients usually occur in the direction normal to the surface, which requires cells of very large aspect ratio. Structured grids are superior in capturing the directionality of the flow field in these viscous regions. A compromise between the two different types of meshes is the use of prismatic grids [2].

Prismatic meshes consist of triangular faces that cover the body surface, while quadrilateral faces extend in the direction normal to the surface. The cells can have very high aspect ratio, while providing geometric flexibility in the lateral directions. The memory requirement of the prismatic grid is much less than that of a tetrahedral grid due to its semi-structured nature. Additionally, the semi-structured nature of the prismatic grid allows simple implementation of algebraic turbulence models and directional multigrid convergence acceleration algorithms [3]. While prismatic meshes are effective in

*Graduate Research Assistant
†Associate Professor

capturing viscous effects in the boundary layers, they cannot cover domains that are multiply-connected. Tetrahedral elements appear to be appropriate to fill the irregular gaps in between the prismatic regions. Their triangular faces can match the corresponding triangular faces of the prisms [4,5].

The initial meshes used for simulation of viscous flows may not always result in the desired accuracy so improvement of the mesh may be necessary. Adaptive grid methods have evolved as an efficient tool to obtain numerical solutions without *a priori* knowledge of the nature and resolution of the grid necessary to efficiently capture the flow features. These algorithms detect the regions that have prominent flow *features* and increase the grid resolution locally in such areas [3,6].

One of the main issues for implementing a Navier-Stokes solver on a parallel computer is partitioning of the complex 3-D computational domain in such a way that the load is balanced among the processors. This task is by no means trivial because the partitioning should result in subdomains that have a balanced load while minimizing the amount of communication required between processors. Many approaches for partitioning of complex computational domains have been developed [7–10]. Two of the more popular techniques are orthogonal recursive bisection and eigenvalue recursive bisection. Orthogonal recursive bisection uses cutting planes to partition the grid based on the centroidal coordinates of the cells. This approach is very inexpensive but the number of elements on the partition interfaces can be large. Eigenvalue recursive bisection requires solution of eigenvalue problems and is quite expensive but this technique reduces the number of elements on partition interfaces [9,10]. One of the more popular eigenvalue techniques is recursive spectral bisection (RSB) [10]. This technique has been used for partitioning of unstructured meshes to obtain near-optimal subdomains.

The current work presents a parallel Navier-Stokes solver and grid adapter along with a new partitioning scheme in which the orthogonal recursive bisection approach is applied to the octree of a hybrid mesh instead of the cells. The resulting subdomains based on the octree have fewer elements on the partition interfaces than if bisection of the cells were performed. The use of the octree also requires less computational time to partition a mesh because the number of octants is significantly smaller than the number of grid cells.

2. Adaptive Numerical Method

The solver employs a finite volume method using the conservative Navier-Stokes equations in integral form. The evaluation of the finite volume integrals is performed on the edges in the mesh to reduce computational time and memory requirements. A Lax-Wendroff approach is applied to march the solution in time [3].

Three-dimensional Navier-Stokes computations usually require a large amount of memory. In the present work, the structure of the prismatic grid in one of the directions is exploited in order to reduce storage to the amount required for a two-dimensional Navier-Stokes solver with triangles. All pointers that are employed refer to the triangular faces of the first prisms on the body surface (*base* faces), with all prisms above the same *base* face forming a stack. A simple index (typical of structured grids indexing) is sufficient to refer to any prism cell belonging to the same stack. The *base* faces pointers connect the

faces to their corresponding edges and nodes. A complete explanation of the solver and its parallel implementation can be found in references [3] and [11].

A special type of adaptive refinement of prisms is applied in the present work in order to preserve the structure of the mesh along the *normal-to-surface* direction. If there is appreciable local flow variation parallel to the surface, then all prisms in the local stacks are directionally divided along the lateral directions. In other words, the triangular faces are divided while the quadrilateral faces are not. In this way, grid interfaces within the prisms region are avoided and the structure of the grid along the *normal-to-the-surface* direction is preserved. Therefore, adaptation of the prisms reduces to adaptation of the triangular grid on the surface resulting in a simpler and less expensive algorithm in terms of storage and CPU time compared to a 3-D adaptation algorithm.

Two types of division are applied. The first divides the triangular faces of the prisms into four smaller triangles, while the second type divides them into two. If two edges of the triangle are flagged for refinement, the third is also flagged automatically to avoid stretching. Edges are chosen for refinement by determining the flow gradients along an edge and if the gradient is above a user-defined threshold, the edge is flagged for refinement. In order to avoid stretched meshes created due to several refinements of the same cells, the following rules are implemented: (i) only one level of refinement/coarsening is allowed during each adaptation, (ii) if the *parent* cell is refined according to one-edge division, then it is divided according to the three-edge division at the next refinement, and (iii) if the maximum adaptation level difference of neighboring surface triangles around a node is more than one, the coarsest triangles will be refined according to the three-edge division.

The adaptation procedure for tetrahedra is very similar to that for prisms. The feature detector flags edges to be refined. The method of tetrahedral cell adaptation employed in the present work is discussed in detail in [6]. The following three types of tetrahedral cell division are considered: (i) one edge is refined, (ii) three edges on the same face are refined, and (iii) all six edges are refined.

After all edges for refinement are flagged, each tetrahedral cell is visited and the flagged edges are counted. Then, the cell is chosen for division according to the above three types. In all cases that are different from the three cases above, the cell is divided according to the third type of division. If two edges on the same face are flagged, the third edge of that face is also marked. In order to avoid a stretched mesh, the previous rules applied to prisms adaptation are employed. A more detailed description of the hybrid grid adaptation algorithm is given in [3].

3. Octree-Based Partitioning of Hybrid Grids

Partitioning of an unstructured mesh is accomplished by using its corresponding octree. The octree is automatically generated by recursive subdivision of a master hexahedron (octant) enclosing the entire domain into successively smaller octants. A sweep over the cells in the domain is performed and the cell is placed in the octant in which its centroid lies. When the number of cells in an octant exceeds a user-specified amount, the octant is refined into eight smaller octants and the cells that were in the parent octant are placed in the appropriate child octant. This process continues until all cells in

the domain are placed in their respective octants. The resulting octree has significantly fewer octants than the total number of cells. The octree also results in a structure that follows the geometry of interest. The computational cells are usually clustered around the body therefore the octants are refined more in this region while the octants near the far field remain relatively large. The computational grid is divided into as many subgrids as processors by a partitioning algorithm. The main steps of the process are (i) coordinate-based grouping of octants and (ii) smoothing of partition boundaries.

3.1. Coordinate-based Grouping of Octants

The grid is partitioned by dividing up the corresponding octree and assigning the cells in an octant to the appropriate subdomain. The octants are divided into groups based upon their centroidal coordinates by cutting planes that are provided as input. By suitably setting the orientation of the cutting planes, several different partitionings of the grid can be realized. The coordinate-based cutting planes are better suited for division of an octree than for partitioning of the computational cells. Division of the computational cells can result in long and slender partitions that have a large percentage of faces, edges, and nodes on the partition interfaces. Since the octree is biased by the size and number of computational cells in a region, the partitions generated by dividing the octants are not as long and slender and they have a lower percentage of grid elements on the partition interfaces.

One feature of the present partitioning method is that a stack of prisms is not split among partitions. The *normal-to-the-surface* structure of the prismatic region is exploited so that all cells within each prism-stack are assigned to the same partition. In this way, all data structure operations for partitioning refer to the triangular surface mesh. This results in savings in both memory and execution time.

3.2. Smoothing of Interpartition Boundaries

Because of the unstructured nature of the grid, interpartition boundaries may be jagged with a high number of edges on partition interfaces. These boundaries can be improved by applying a smoothing technique to the cells on the partition interfaces. The process begins by determining which cells in the computational domain are candidates for smoothing. A cell is flagged for smoothing if all of its nodes are shared between two neighboring partitions. Thus, if the cell were moved from its present subdomain to the neighboring one, then the number of grid elements on the interface boundary would be reduced. The flagged cells are then moved among the neighboring subdomains so that the number of grid elements on the interface is reduced while maintaining a load balance among the partitions. This process repeats iteratively until almost all of the flagged cells on the interfaces have been smoothed. This smoothing has proven to reduce the percentage of grid elements on the partition interfaces and the process usually completes in less than ten iterations.

4. Results for an Aircraft Configuration

The performances of the coordinate-based octant grouping scheme, parallel solver, and parallel grid adapter are now examined for simulation of flow about a high speed civil transport (HSCT) aircraft configuration. The initial mesh consists of approximately 176K

prisms and 170K tetrahedra. The corresponding octree for this mesh contains just over 15K octants. Figure 1 shows the signature of the partitions on the surface of the HSCT after smoothing of the interpartition boundaries for a case with sixteen prism partitions. The partition interfaces are smooth with approximately ten percent fewer edges on the boundaries than if smoothing were not applied. The partitioning of the corresponding octree for the same case with sixteen tetrahedral partitions is shown in Figure 2a. The figure shows the footprint of the partitions on the symmetry plane of the domain. The octree biases the partitioning around the aircraft geometry. The footprint of the resulting tetrahedral subdomains on the symmetry plane after smoothing of partition boundaries is shown in Figure 2b.

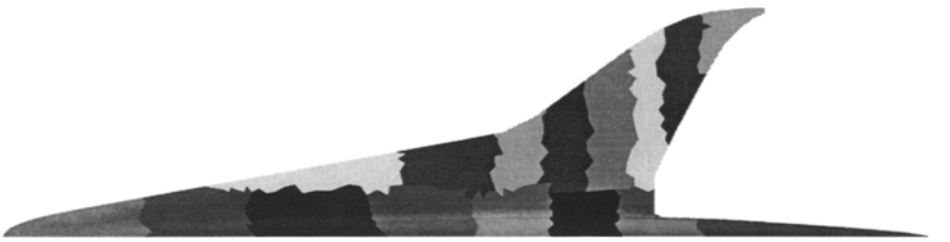


Figure 1. Signature of the partitions on the surface of the HSCT aircraft after smoothing of the interpartition boundaries for a case with sixteen prism partitions.

Figures 3a and 3b compare the maximum local percentage of edges on partition interfaces for the current octree partitioning and a partitioning generated using RSB. The octree technique yielded a slightly higher percentage of edges on the boundaries than the RSB method. The same trend was observed for the average local percentage of edges on partition interfaces with the RSB performing only slightly better than the octree method. It should be emphasized, however, that the execution time for the present partitioning technique is much less than the amount of time required by RSB to partition the same grid.

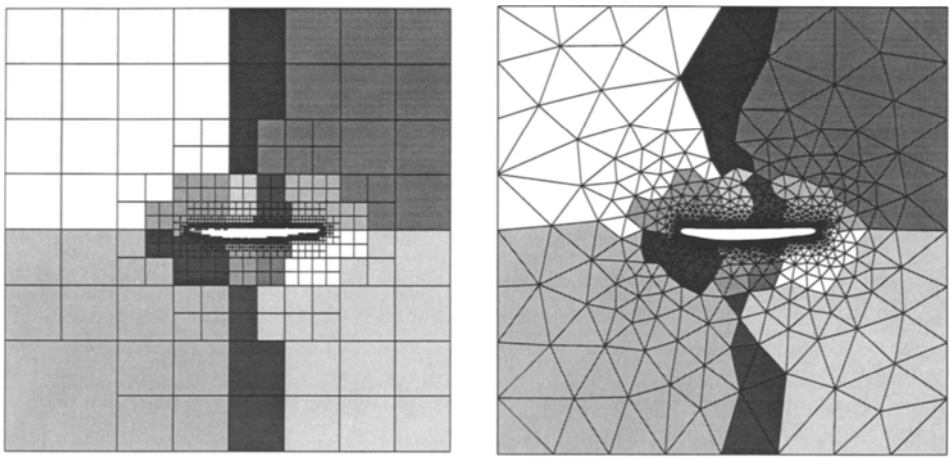


Figure 2. Partitioning of the octree corresponding to the HSCT tetrahedral mesh using coordinate-based cutting planes results in subdomains that are biased by the geometry. Signatures of the sixteen partitions are shown on the symmetry plane for both (a) the octree and (b) the tetrahedral mesh after smoothing of partition boundaries.

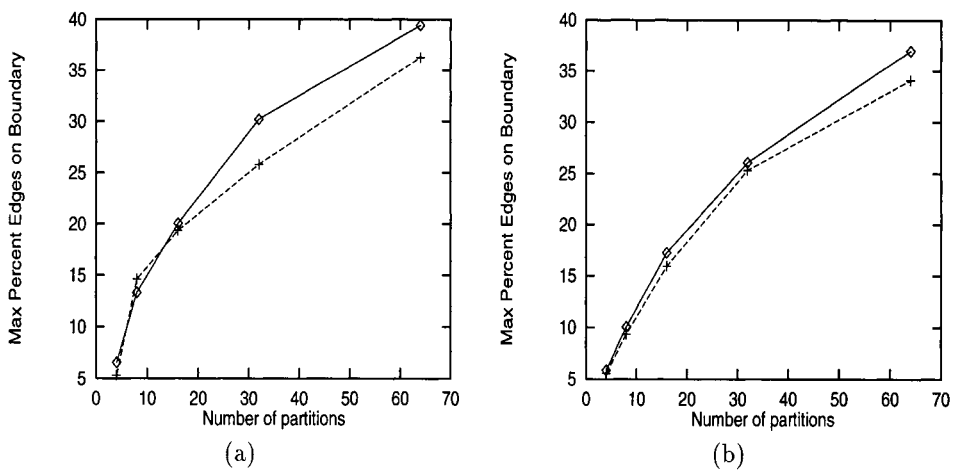


Figure 3. Comparison of the maximum local percentage of edges on partition interfaces obtained using the coordinate-based octree method and RSB for the (a) prismatic and (b) tetrahedral regions around the HSCT.
-◇- using octree-based partitioning, -+- using recursive spectral bisection (RSB).

Several of the partitions obtained using the octree method were run using the solver and then adapted on an Intel Paragon. Figure 4a presents the scalability results for parallel execution of the solver for the HSCT. The scale on the axes is logarithmic with base 2. The execution times per time step for increasing number of processors exhibit a linear reduction with only a slight deviation from the ideal linear reduction in time. It is noted that the communication times for the solver were less than one percent of the total run time for all cases considered. The scalability results for parallel execution of the grid adapter for the same grid are shown in Figure 4b. A dynamic load balancer used in conjunction with the presented parallel grid adapter to rebalance the loads among processors after adaptation is given in reference [12].

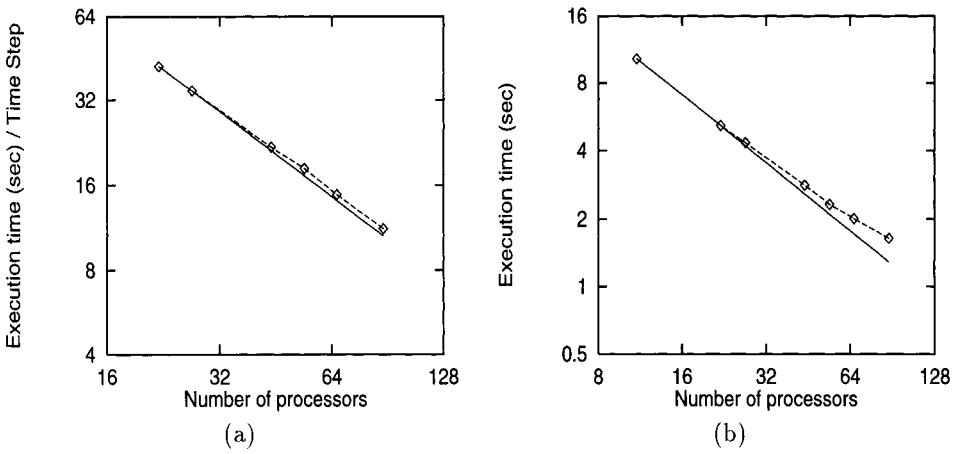


Figure 4. Scalability results for parallel execution of the (a) solver and (b) grid adapter on the Intel Paragon for the HSCT aircraft configuration. Case of hybrid grid with 176,480 prisms and 170,300 tetrahedra.
-◇- actual time, — ideal linear reduction in time.

Acknowledgments

The authors would like to thank Horst Simon for providing us with the RSB partitioning code Version 2.2 written by Steve Barnard and Horst Simon. This work was supported by ARPA Grant DABT 63-92-0042, the Texas Advanced Technology Program (ATP) Grant # 003658-413, and the NSF Grant ASC-9357677 (NYI program). Parallel computing time on the Intel Paragon was provided by the NAS Division of the NASA Ames Research Center.

REFERENCES

1. T.J. Baker, "Developments and Trends in Three Dimensional Mesh Generation," *Applied Numerical Mathematics*, Vol. 5, pp. 275-304, 1989.
2. Y. Kallinderis and S. Ward, "Prismatic Grid Generation for 3-D Complex Geometries," *AIAA Journal*, Vol. 31, No. 10, pp. 1850-1856, October 1993.
3. V. Parthasarathy, Y. Kallinderis, and K. Nakajima, "Hybrid Adaptation Method and Directional Viscous Multigrid with Prismatic / Tetrahedral Meshes," AIAA Paper 95-0670, Reno, NV, January 1995.
4. Y. Kallinderis, A. Khawaja, and H. McMorris, "Hybrid Prismatic / Tetrahedral Grid Generation for Complex Geometries," AIAA Paper 95-0211, Reno, NV, January 1995.
5. A. Khawaja, H. McMorris, and Y. Kallinderis, "Hybrid Grids for Viscous Flows around Complex 3-D Geometries including Multiple Bodies," AIAA Paper 95-1685-CP, San Diego, CA, June 1995.
6. Y. Kallinderis and P. Vijayan, "An Adaptive Refinement Coarsening Scheme for 3-D Unstructured Meshes," *AIAA Journal*, Vol 31, No.8, pp 1440-1447, August 1993.
7. C. Farhat and M. Lesoinne, "Automatic Partitioning of Unstructured Meshes for the Parallel Solution of Problems in Computational Mechanics," *International Journal for Numerical Methods in Engineering*, Vol. 36, pp. 745-764.
8. R. Lohner, R. Ramamurti, and D. Martin, "A Parallelizable Load Balancing Algorithm," AIAA Paper 93-0061, Reno, NV, January 1993.
9. E. R. Barnes, "An Algorithm for Partitioning the Nodes of a Graph," *SIAM Journal of Alg. Disc. Methods*, Vol. 3, p. 541, March 1982.
10. A. Pothen, H. D. Simon, and K.-P. Liou, "Partitioning Sparse Matrices with Eigenvectors of Graphs," *SIAM Journal of Matrix Anal. Applications*, Vol. 11, No. 3, pp. 430-452, July 1990.
11. T. Minyard and Y. Kallinderis, "A Parallel Navier-Stokes Method and Grid Adapter with Hybrid Prismatic/Tetrahedral Grids," AIAA Paper 95-0222, Reno, NV, January 1995.
12. T. Minyard and Y. Kallinderis, "Partitioning and Dynamic Load Balancing of Adaptive Hybrid Grids," AIAA Paper 96-0295, Reno, NV, January 1996.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel Adaptive *hp* Finite Element Approximations For Stokesian Flows: Adaptive Strategies, Load Balancing and Domain Decomposition Solvers

Abani Patra*and J. T. Oden†
Texas Institute For Computational and Applied Mathematics
University of Texas at Austin
Austin,TX-78712

Abstract

This paper summarizes the development of a new class of algorithms using parallel adaptive *hp* finite elements for the analysis of Stokesian flows. Adaptive strategies, mesh partitioning algorithms and a domain decomposition solver for such problems are discussed.

1 Introduction

In this paper, we describe some new algorithms and early experiences with them, in combining adaptive *hp* finite element methods with parallel computing for the analysis of Stokesian flows. Adaptive *hp* finite element methods, in which grid size and local polynomial order of approximation are both independently adapted, are capable of delivering super-algebraic and even exponential rates of convergence, as seen in the work of Babuska, Oden, Demkowicz and others [1, 2]. With parallel computing, these methods have the potential to dramatically reduce computational costs associated with realistic finite element approximations.

The development of several good *a posteriori* estimators [3, 4] has removed one of the principal difficulties in implementing *hp* adaptive methods. However many other difficulties must be surmounted before such performance can be achieved for real simulations. Major difficulties in doing so are: non-conventional adaptive strategies that produce good *hp* meshes must be developed; the linear systems arising out of non-uniform *hp* meshes are difficult to partition into load balanced sub-domains and are very poorly conditioned for efficient parallel iterative solution. The subsequent sections describe algorithms designed to overcome each of these difficulties. Section 2 introduces the Stokes problem its' finite element formulation and

*NSF Post-doctoral Fellow, TICAM
†Cockrell Family Regents Chair no. 2 in Engineering

the appropriate function spaces necessary for a description of the various algorthms used in the solution process. Section 3 describes a simple adaptive strategy for producing good *hp* meshes. We also discuss here a construction of compatible approximation spaces for the velocity and pressure spaces. Section 4 reviews a recursive load based bisection type mesh partitioning strategy. Section 5 discusses a domain decomposition type solver for such problems.

2 The Stokes Problem

The mixed finite element approximation of the Stokes problem is classically given by:

Find $\boldsymbol{u}_h \in V_h$, $P_h \in W_h$ such that

$$a(\boldsymbol{u}_h, \boldsymbol{v}) + b(\boldsymbol{v}, P_h) = L(\boldsymbol{v}) \qquad \forall \boldsymbol{v} \in V_h \tag{1}$$

$$b(\boldsymbol{u}_h, q_h) = -b(\bar{\boldsymbol{u}}, q_h) \qquad \forall q_h \in W_{h0} \tag{2}$$

where $\boldsymbol{u}_h + \bar{\boldsymbol{u}}$ is the approximate velocity field and P_h is the approximate pressure field of an incompressible viscous fluid flowing through a given domain $\Omega \subset \mathbb{R}^d$ with imposed velocity $\bar{\boldsymbol{u}}$ at the boundary $\partial\Omega$ of Ω. The domain Ω is partitioned into sub-domains Ω_i and finite elements ω_K. It is assumed that sub-domain boundaries coincide with element boundaries. The finite element spaces V_h and W_{h0} are conforming finite dimensional approximations of $(H_0^1(\Omega))^2$ and $L_0^2(\Omega)$.

$$V_h = \{\boldsymbol{v}_h \in V^{hp}, \boldsymbol{v}_{h|\omega_K} \in V_p(\omega_K), \forall K, \boldsymbol{v}_h = 0 \text{ on } \partial\Omega\}$$

$$W_h = \{q_h \in L^2(\Omega), q_{h|\omega_K} \in W_p(\omega_K), \forall K\},$$

$$W_{h0} = \{q_h \in W_h, \int_\Omega q_h d\boldsymbol{x} = 0\}$$

$V_p(\omega_K)$ and $W_p(\omega_K)$ are tensor product polynomial spaces whose precise construction is described in the next section.

3 3 Step Adaptive Strategy

This type of adaptive strategy was first proposed in Oden and Patra [6]. We briefly review the underlying ideas. Most conventional adaptive strategies propose an *incremental* type of refinement whereby a certain heuristically determined fraction of the mesh is refined/enriched to the next level. While this strategy ultimately leads to a good mesh, it causes a large number intermediate solution steps on non-optimal meshes. The cost of these intermediate solutions might negate all advantages of adaptivity. Clearly a good adaptive strategy must deliver a mesh for a desired level of error in one or two iterations. Moreover, in the context of *hp* adaptivity, we also need a way of choosing between *h* and *p* adaptivity.

3.1 Basic Principles and Strategy

We now describe the basic ideas underlying the adaptive strategy.

- Optimal meshes *equidistribute error* over the whole domain.
- Asymptotic *a priori* error bounds are treated as *equalities* e.g.

$$||e||_{1,\Omega} = C \sum_{K=1}^{N} \frac{h_K^{\mu}}{p_K^{\nu}} ||u||_{r,\Omega_K} = \sum_{K=1}^{N} \frac{h_K^{\mu}}{p_K^{\nu}} \Lambda_K$$

  where u is the exact solution, e is the error, h and p are mesh parameters.

- Use of a good *a posteriori* error estimate to compute the constants Λ_K (mesh parameter independent) in the above *a priori* error estimate from a coarse mesh solution. Convergence rates μ and ν, if unknown, can be estimated from two coarse mesh solutions.
- Mesh parameters h and p, required for a desired error, are estimated *locally* from the *a priori* estimate and the need to *equidistribute the error.*
- Orthogonality of error to finite element space leads to a good approximation of the norm of the exact solution norm i.e. $||u||_{1,\Omega}^2 \approx ||u^{hp}||_{1,\Omega}^2 + ||e||_{1,\Omega}^2$

The adaptive strategy comprises of 3 steps: 1) selecting an intermediate error level between the initial mesh error and the final target mesh, and estimating different parameters using a coarse mesh solution, 2) keeping polynomial orders p_K constant adapting the grid size (change h_K) to achieve the intermediate error while equidistributing the error, 3) keeping grid size constant and changing the local polynomial orders p_K to achieve the target error while equidistributing the error.

3.2 Compatible approximations of velocity and pressure

We will now specify the exact polynomial spaces $V_p(\omega_K)$ and $W_p(\omega_K)$ that avoid the commonly known phenomena of "mesh locking" while preserving approximation properties in an adaptive hp mesh. In mathematical terms, these spaces satisfy the requirement that the LBB constant is bounded away from zero. Our ideas here are motivated by the work of Stenberg and Suri [5] on the p version.

Let $U_i(x) = \int_{-1}^{x} L_i(t)dt$, where $L_i(x)$ is the Legendre polynomial of degree i and let $P_p(S)$ denote a polynomial of degree totaling p defined on the unit square $S = [-1,1]^2$. Now the approximation over each element ω_K may be defined by the sum of internal functions $J_p(S)$ and external functions $E_p(S)$ defined as

$$J_p(S) = \{v | v = \textstyle\sum_{i,j=1}^{p-1} a_{ij} U_i(x) U_j(y), a_{ij} \in \mathbb{R} \, p \geq 2\}$$
$$E_p(S) = P_1(\hat{I}_x) P_p(\hat{I}_y) \cup P_p(\hat{I}_x) P_1(\hat{I}_y)$$

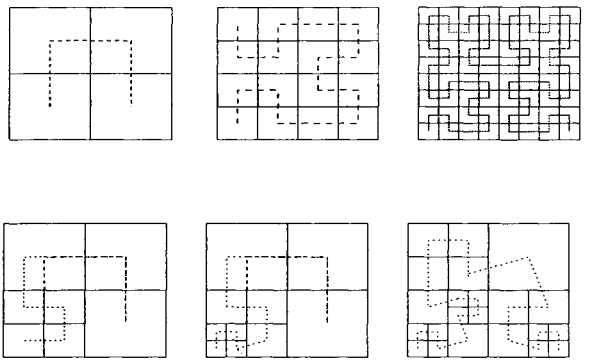


Figure 1: Basic space filling curves.

where I_x and I_y are the unit intervals $[-1,1]$. Thus in two dimensions E_p is made up of four sets of polynomials associated with the four edges of S. In an adaptive hp mesh these may all be different. Now let $p_m = max\{p_1, p_2, p_3, p_4\}$. Compatible spaces $V_p(S)$ and $W_p(S)$ can be defined as

$$W_p(S) = E_{p1-1}(\gamma_1) \oplus E_{p2-1}(\gamma_2) \oplus E_{p3-1}(\gamma_3) \oplus E_{p4-1}(\gamma_4) \oplus J_{p_m-1}(S)$$

$$V_p(S) = E_{p1}(\gamma_1) \oplus E_{p2}(\gamma_2) \oplus E_{p3}(\gamma_3) \oplus E_{p4}(\gamma_4) \oplus J_{p_m+1}(S).$$

where γ_1, γ_2 etc. denote the sides of S The standard finite element mapping process can be used to obtain $V_p(\omega_K)$ and $W_p(\omega_K)$.

4 RLBBO – Mesh Partitioning Algorithm

An essential part of applying parallel computation to these problems is the partitioning of the mesh into load balanced pieces with minimal interfaces. Partitioning adaptive hp meshes, however, poses special difficulties since i)the load distribution is irregular and localized ii) a good choice of an *a priori* measure of computational load is difficult.

We use a recursive bisection of an ordering of the elements created using a space filling curve (see Fig. 1 for an illustration) passing through the element centroids. The curve is bisected using a composite load measure comprising of the load estimates on each element in a partition and the engendered interface. As a load measure we use the degrees of freedom in each element, the error in a coarser mesh and the degrees of freedom on interfaces. Such algorithms are discussed in Patra and Oden [7].

5 Domain Decomposition Solver

The solver described here is an extension of the domain decomposition solver proposed for adaptive *hp* methods for elliptic problems in Oden, Patra and Feng [8]. The primary concerns with this type of solver are: 1) is the preconditioner good enough to guarantee convergence with increasing p, and 2) is the solver efficient for parallel computing. The solution process for Stokesian flows poses two additional difficulties: 1) obtaining solution in *divergence free space*, 2) the solution of *indefinite linear systems*. The basic solution procedure for elliptic problems covered by the following scheme:

Apply partial orthogonalization first at the element level to eliminate the interior functions and then at the sub-domain level to obtain reduced system on the interface $S\bar{u} = F$, where $S = \sum_i^{N_D} S_i$ and $S_i = \sum M_i^T K_i M_i$ and $F = \sum F_i = \sum M^T f_i$

- $R^0 = F$, $P^0 = 0$
- Iteration in n
  - Precondition $G^n = C^{-1} R^n$
  - Compute direction of descent

$$\mathbf{P^n} = \mathbf{G^n} + \frac{< R^n, G^n >}{< R^{n-1}, G^{n-1} >} P^{n-1}$$

  - Compute

$$Z^n = \sum_{i=1}^{N_D} S_i P^n, \alpha_n = \frac{< R^n, G^n >}{< Z^n, P^n >}$$

$$\mathbf{u^{n+1}} = \mathbf{u^n} + \alpha_\mathbf{n} \mathbf{P^n}, R^{n+1} = R^n - \alpha_n Z^n$$

- end loop on n
- Add correction term to interior unknowns $u_i = u_i^0 + M_i^T \bar{u}$

5.1 Solution in divergence free space

It is clear fron the above algorithm that to construct divergence free velocities by the above procedure, one must ensure that each of the search direction P^n is a divergence free vector. This is accomplished by modifying the preconditioning step $\mathbf{C}\ G^n = R^n$ to

$$\begin{aligned} CG^n + \quad \bar{B}^T \bar{p} &= g \\ \bar{B} G^n &= 0 \end{aligned}$$

where

$$\bar{B} = \int_{\Omega_i} \nabla v_h . 1 d\Omega_i = \int_{\Gamma_i} v_h . n d\Gamma$$

and $\bar{p}$ is a vector of average pressure per sub-domain. This computation reduces to one coarse solveper iteration of a problem of dimension equal to the number of sub-domains, and the initial cost of setting up and factoring $\bar{B}C^{-1}B^T$.

5.2 Solution of Indefinite Systems

If the pressures are discontinuous across inter-element boundaries: then they may be eliminated at the element level using one more step of partial orthogonalization as shown below. Consider the element matrix

$$[K_e]\begin{Bmatrix} u_e \\ P \\ u_b \end{Bmatrix} = \begin{bmatrix} K_{ee} & B_{eP} & K_{eb} \\ B_{Pe} & 0 & B_{pb} \\ K_{be} & B_{bp} & K_{bb} \end{bmatrix}\begin{Bmatrix} u_e \\ P \\ u_b \end{Bmatrix}$$

Apply partial orthogonalization

$$\left[\widehat{K_e}\right]\begin{Bmatrix} \widehat{u_e} \\ \widehat{P} \\ u_b \end{Bmatrix} = \begin{bmatrix} \widehat{K_{ee}} & \widehat{B_{eP}} & 0 \\ \widehat{B_{Pe}} & S & 0 \\ 0 & 0 & K_{bb} \end{bmatrix}\begin{Bmatrix} \widehat{u_e} \\ \widehat{P} \\ u_b \end{Bmatrix}$$

where $S = -B_{pb}K_{bb}^{-1}B_{pb}$.

Following up with one more level of partial orthogonalization

$$\left[\widetilde{K_e}\right]\begin{Bmatrix} \widetilde{u_e} \\ \widetilde{P} \\ u_b \end{Bmatrix} = \begin{bmatrix} \widetilde{K_{ee}} & 0 & 0 \\ 0 & \widetilde{S} & 0 \\ 0 & 0 & K_{bb} \end{bmatrix}\begin{Bmatrix} \widetilde{u_e} \\ \widetilde{P} \\ u_b \end{Bmatrix}$$

Element average pressures $\bar{p_e}$ can be recovered by postprocessing the original equations with the velocities. Thus the final pressure is formed as $P = \bar{p_i} + \bar{p_e} + \tilde{P}$.

5.3 Choice of Preconditioner

As described in Oden, Patra, and Feng [8] matrix S is naturally blocked into a small portion (NN) corresponding to the linear on the interface and the larger portion corresponding to the unknowns associated with the higher order polynomials(EE) and their interactions NE and EN. As a preconditioner C, we explore two choices, denoted HPP1 and HPP2, respectively: 1) the NN block and the diagonals of EE 2) the NN block and the block diagonals of EE corresponding to a particular edge. For these choices of preconditioner, Letallec and Patra [9] have established firm theoretical bounds on the conditioning of the system, guranteeing convergence in a reasonable number of iterations.

6 Numerical Results

Example 1. The well known problem of driven cavity flow is used as the first numerical example. The domain, boundary conditions and mesh are shown in Fig. 2. Figure 3 shows performance of the domain decomposition solver with respect to increasing polynomial order and problem size. Convergence with respect to both appears to be robust and correspond to the theoretical bounds. Page limitations prevent inclusion of further results here.

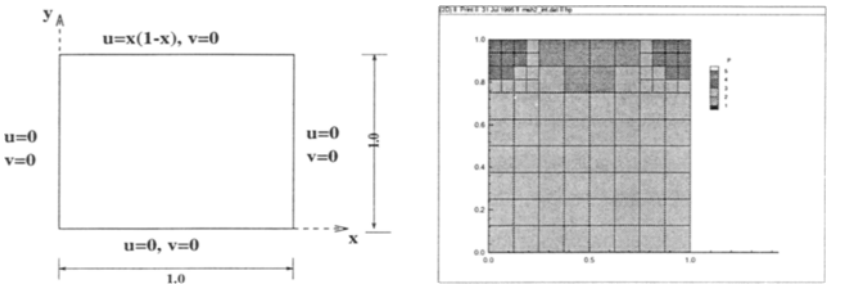


Figure 2. a)Driven cavity flow – boundary conditions and domain b)Sample adaptive *hp* mesh

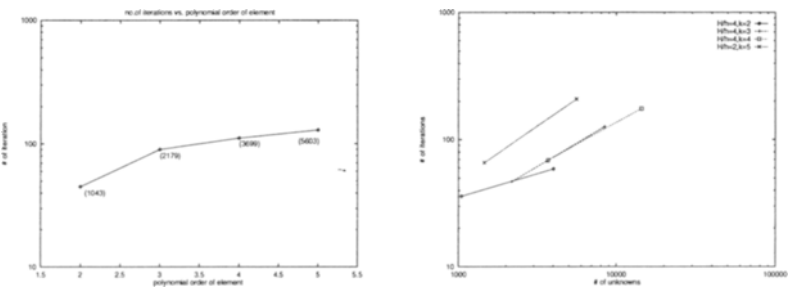


Figure 3.a)Convergence of iterative solver with HPP1 preconditioner for different element orders. Mesh of 64 elements with H/h=4. Numbers in parentheses are the total number of unknowns. b)Scalability of iterative solver with HPP1 preconditioner for different element orders. Effect of problem size keeping number of elements per processor constant for different element orders.

Acknowledgements

The support of this work by ARPA under contract no. DABT63-92-C-0042 is gratefully acknowledged.

References

[1] I. Babuska and M. Suri, "The p and h-p versions of the finite element method, Basic Principles and Properties", *SIAM Review*, Vol. 36, Number 4, December 1994.

[2] J. T. Oden and L. Demkowicz, " h-p adaptive finte element methods in computational fluid dynamics", *Computer Methods in Applied Mechanics and Engineering*, Vol. 89, 1991.

[3] M. Ainsworth , J. T. Oden, "A Unified Approach to *a-posteriori* Error Estimation Using Element Residual Methods " ,*Numerische Mathematik*, vol. 65 (1993) pp.23-50.

[4] R. E. Bank, R. K. Smith, "*A posteriori* Error estimates based on hierarchical bases", *SIAM Journal on Numerical Analysis*, vol 30 no. 4, pp. 921-935.

[5] R. Stenberg and M. Suri, "Mixed hp finite element methods for problems in elasticity and Stokes flow", preprint, February 1994.

[6] J.T. Oden and Abani Patra, "A Parallel Adaptive Strategy For *hp* Finite Elements",in *Comp. Meth. in App. Mech. and Engg.*, vol 121, March 1995, pp. 449-470.

[7] Abani Patra and J. T. Oden "Problem Decomposition Strategies for Adaptive *hp* Finite Element Methods", to appear in *Computing Systems in Engineering.*

[8] J. T. Oden, Abani Patra, Y. S. Feng, ""Domain Decomposition for Adaptive *hp* Finite Elements", in J. Xu and D. Keyes ed. **Procedings of VII th International Conference on Domain Decomposition Methods**, State College, Pennsylvania.

[9] P. Letallec and A. K. Patra, "Non-overlapping Domain decomposition Methods for Stokes Problems with Discontinuous pressure fields", TICAM Report (in press).

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel Processing for Solution-Adaptive Computation of Fluid Flow *

T. L. Tysinger, D. Banerjee, M. Missaghi and J. Y. Murthy

Fluent Inc.
10 Cavendish Court
Lebanon, NH 03766
USA

This paper describes the parallel implementation of a solution-adaption algorithm for the computation of fluid flow on unstructured grids. Comparison to the serial implementation is demonstrated and the performance of the overall parallel adaption procedure is presented.

1. INTRODUCTION

Fluid flow problems involving regions of sharp gradients, multiple scales, or moving fronts are common in a number of engineering application areas such as phase change in casting and molding, pollutant dispersal, ground water transport, combustion, in-cylinder flows in the automotive industry and in countless other applications. In mold-filling, for example, a molten fluid is poured into a mold. The melt free surface moves through the mold cavity with time, displacing the air; as it moves, it may also change phase as the cold mold walls cause multiple freeze fronts to move inwards. Such problems generally involve complex three-dimensional geometries, are time-dependent, and consequently, extremely computer-intensive. Parallel processing offers an effective option for solving these problems in a timely manner.

Moreover, virtually all problems of industrial interest involve complex geometries, and much of the user's time is consumed in geometry and mesh generation. In recent years, unstructured mesh methodologies have been developed which offer great flexibility in discretizing complex domains, and in dynamically adapting the mesh to the evolving solution. Unstructured solution-adaptive meshes allow optimal use of computational resources because grid nodes may be concentrated dynamically in areas of high gradients, without the topological constraints imposed by line or block mesh structure.

The overall objective of our efforts is to develop efficient and accurate computational methods for solving moving front problems on unstructured meshes, utilizing solution adaption to capture the moving front and parallel processing to achieve computational speed. This paper presents our progress to date in developing the necessary procedures for

*This material is based upon work supported by the National Science Foundation under award number DMI-9360521. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

parallel solution adaption, partitioning, and load balancing in the context of unstructured meshes.

2. APPROACH

2.1. Numerical Scheme

The flow solver is based on a finite volume scheme wherein the physical domain is subdivided into small volumes, or cells, and the integral equations are applied to each cell. A central difference scheme is used to compute convective and diffusive fluxes. Artificial dissipation is therefore required to stabilize the solution. Several options are available for computing this, including a blend of second and fourth differences of conserved variables similar to that suggested by Jameson et al [1]. Alternatively, an upwinded flux evaluation scheme, such as the Roe flux function [2], may be employed that does not require any additional dissipation. Instead of using the cell-average values of the conserved variables as the left and right states for computing the face fluxes, a multidimensional linear reconstruction approach similar to that suggested by Barth and Jespersen [3] is used to increase the spatial accuracy of the discretization. The basic time stepping mechanism is a user-configurable explicit multi-stage Runge-Kutta scheme, with either a global time step for time-accurate solutions or local time stepping for faster convergence to steady state solutions. To enhance convergence rates, a novel multigrid scheme [4] that employs a sequence of nested coarse meshes made up of faces of the fine mesh is used. The multigrid scheme can be coupled with a dual-time stepping method [5] for the solution of unsteady flows. Preconditioning algorithms are used to overcome time-step restrictions encountered for low Mach number or incompressible flows [6].

The parallel implementation of this solver is described in [7]. It is based on a domain decomposition strategy in which the spatial domain for a problem is separated into smaller sub-domains or partitions and a separate instance of the solver is invoked to simulate the flow within each partition.

2.2. Solution Adaption

The parallel adaption strategy is an extension of the serial algorithm developed by Smith and demonstrated in [8]. As with the serial implementation, initial cells are identified and marked for adaption based on a general user-selectable criterion (for example, pressure gradient). Once marked, an instance of the serial adaption algorithm is run within each partition (i.e. processor), and message-passing is performed when necessary (e.g. when cells along an interface boundary are refined or coarsened) to perform remote updates and make remote requests.

The refinement algorithm selects a marked cell and splits it along its longest edge. In order to maintain good grid quality for the solver, the algorithm cannot create small cells adjacent to large ones. To satisfy this requirement, the algorithm will search for the cell with the longest edge in the neighborhood of the marked cell and split that edge, thereby reducing the size of neighboring cells. This continues recursively until the initial marked cell is split.

In the parallel implementation each partition will select a marked cell for splitting, if one exists, and apply the above procedure. If the recursive search "falls off" the processor's partition, it is re-started in the neighboring partition. The processor suspends refinement

of the original cell until the neighboring processor splits all large cells of the neighborhood which lie in its partition. In addition, when a cell adjoining an interface is split, the changes must be propagated to all neighboring processors which share a face of the split cell. Since message-passing overhead depends largely on the number of messages sent, several small instruction messages are cached and sent as a single message.

The serial coarsening algorithm determines that a node is to be deleted if all cells adjoining the node are marked for coarsening. The node is deleted and a cavity is constructed from cell faces that form the boundary around marked cells. Next, the cavity is re-triangulated and thereby filled with new and fewer cells. In the parallel implementation, each processor searches for such nodes that can be deleted, and re-triangulates the bounding cavities. Complications arise if the cavity is distributed across more than one processor. In such a case, all cells are migrated to a designated processor which performs the coarsening.

2.3. Load Balancing

Load balancing is important whenever the work (i.e. elapsed wall clock time) required to compute the flow field in any given partition differs significantly from the mean. Without load balancing, those processors which finish "early" will sit idle at synchronous communication points, waiting for the remaining processors to finish their computation. The load balancing procedure is divided into two phases. In the first phase, all cells are assigned a partition destination. The criteria for deciding the cell's destination form the basis for parallel partitioning. In the second phase, cells marked with a new partition destination are sent to that partition. This phase, referred to as *cell migration*, is a general algorithm which allows the migration of any cell to any partition.

2.3.1. Parallel Partitioning

As with the serial algorithms, parallel partitioners attempt to create a number of subdomains (equal to the number of available processors) of approximately equal size with small interface boundaries. The major complication with the parallel algorithms is that the grid geometry and connectivity are distributed among all processors. In this work, for a "proof-of-concept", we have not concentrated on developing or implementing algorithms which produce the "best" interface boundaries, but rather have selected two simple schemes which produce a balanced load with reasonable, but not optimal, interface size.

Global Partitioning

The first partitioning algorithm considered is coordinate strip bisection. Each processor contributes to a global array the centroid location of each cell in its partition. The global array is then sorted from smallest to largest value and sub-divided into the number of processors. Those cells in the first sub-division are assigned a destination of the first processor, the second sub-division are assigned a destination of the second processor, etc. Cells are then migrated to their destination processor if necessary. Global partitioning schemes are often criticized because global reordering and migration of data is required (see for example [9]). However, global reordering can be done efficiently with parallel sorting techniques, and after destinations have been assigned, only one migration pass is needed.

Local Partitioning

Another approach to partitioning is to compute an imbalance between partitions, and exchange cells locally with neighboring partitions. This process continues iteratively until a balanced load is achieved. The process is very similar to solving Laplace's equation for heat diffusion using finite-differences. Here, each partition can be thought of as one node of the computational grid. Nodal temperatures are replaced by the average load in each partition, and differences in the partition loads are computed. Cells are exchanged across interface boundaries in proportion and direction of the gradient. Several local cell migration passes are usually necessary before the load reaches equilibrium. Similar approaches are described by Löhner et al. [9].

2.3.2. Cell Migration

The underlying tool used to load balance is cell migration, where cells are sent to their marked partition. The data structure of the code consists of faces, cells, and nodes. A face contains only pointers to those cells and nodes it adjoins. Copies of faces along an interface boundary reside in both adjacent partitions. Cells, which contain the flow variables, cannot be shared within a partition; however, any residing in an exterior cell layer are duplicates of those in the neighboring partition. Nodes contain geometrical coordinate information, and may be shared by faces within a partition. As with faces, nodes along an interface boundary will be duplicated in the adjacent partitions. Unlike faces, nodes can adjoin more than two partitions.

Cell migration involves four steps:

1. cell transfer: send contents – id and flow variables – of all marked cells to new destination, unless they already exist there; receive incoming cells and add to local list of immigrant cells.

2. node transfer: mark all nodes of marked cells for migration; send contents – id and coordinate geometry – to new destination, unless they already exist there; receive incoming nodes and add to local list of immigrant nodes.

3. face transfer: mark all faces adjacent to the marked cells for migration; send contents – ids of adjacent cells and nodes – to new destination, unless they already exist there; receive incoming faces and add to local list of immigrant faces.

4. rebuild grid: remove all emigrant cells, nodes, and faces from partition if no longer needed; build new section of partition from contents of local immigrant lists.

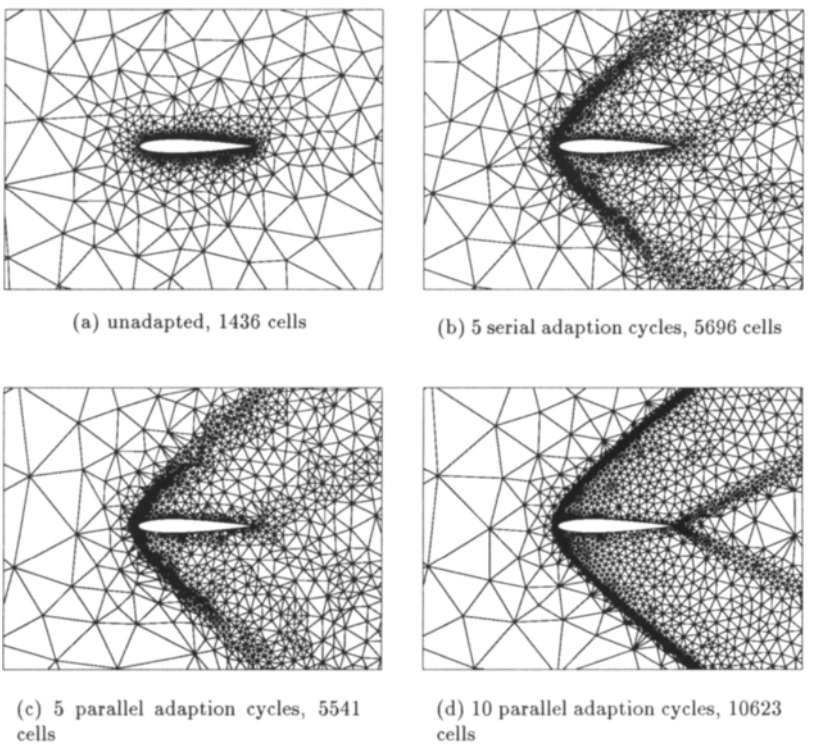
Before the final step, large parts of the domain may be duplicated on the immigrant lists. To reduce this memory overhead, the migration procedure is executed in several passes, limiting the total number of migration cells to a fixed number in each pass. The memory savings are not free since this requires more communication and computation.

3. Results

3.1. Flow Over NACA-0012 Airfoil

To demonstrate the implementation we have devised a problem of supersonic flow over a NACA-0012 airfoil at free stream at Mach number 2. A bow shock develops near the

leading edge together with a shock emanating from the trailing edge. Associated with the shocks are large pressure gradients.



(a) unadapted, 1436 cells

(b) 5 serial adaption cycles, 5696 cells

(c) 5 parallel adaption cycles, 5541 cells

(d) 10 parallel adaption cycles, 10623 cells

Figure 1. Adapted shocks about NACA 0012 airfoil at M=2

The initial grid shown in Figure 1(a) contains 1436 cells but is too coarse to satisfactorily capture the shocks. To resolve the pressure gradients the grid is refined in areas where they are high. A solution is then computed on the newly adapted grid and the process is repeated until the gradients are sufficiently resolved. Figure 1(b) shows the grid after 5 cycles of this adaption process using the serial implementation. The grid at the same number of adaption cycles using the parallel implementation is shown in Figure 1(c). The slight differences between the two grids are due to algorithmic asymmetries in the adaption splitting procedure. Qualitatively there are no differences between the two grids, and the solutions are found to be in agreement. The grid after 10 parallel adaption cycles is shown in Figure 1(d). The grid contains 10623 cells, about 10 times that of the initial grid. To obtain similar resolution on a uniform unstretched grid would have required

more than 2 million cells.

Another important part of the adaption procedure is load balancing. After the grid is adapted, the amount of cells in each partition may vary substantially. For example if the shock happened to reside mostly within one partition, most newly created cells would also reside in that partition. Since the overall speed of the computation is limited by the slowest processor it is desirable that each processor has an equal number of cells. Therefore after each adaption, the grid is re-partitioned in parallel to equally distribute the cells. The grid in Figure 1(c) for example contains 5541 cells. Immediately after it is adapted the number of cells in each partition varies from 921 to 1433. After global repartition, four of the processors contain 1108 cells, and the other 1109 cells. Not considering communication costs, the unbalanced case would require nearly 25% more time to complete.

3.2. Incompressible Flow in a Triangular Driven Cavity

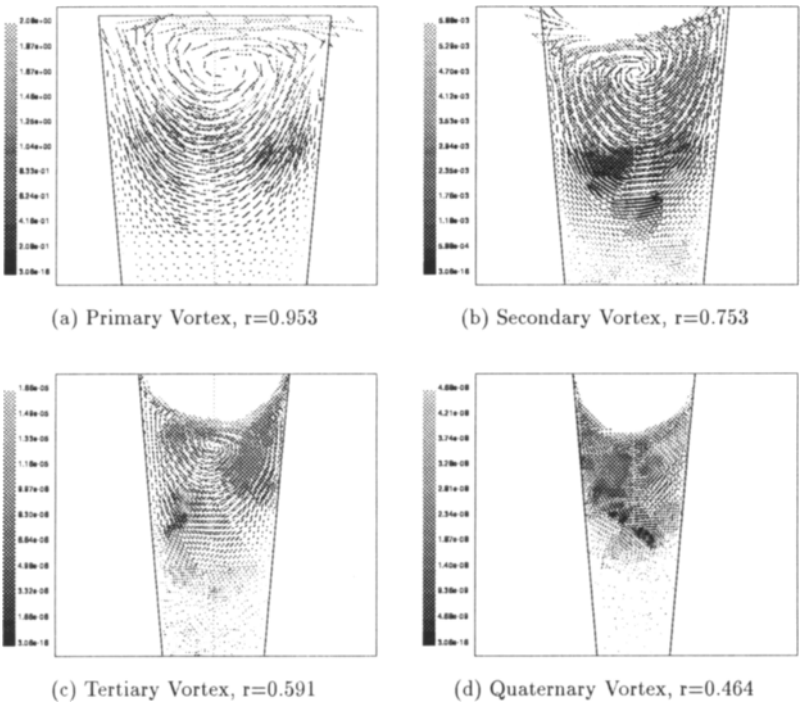
In this study, two-dimensional, steady laminar motion of a fluid within a lid-driven cavity is considered. It is generally agreed that there is a dominant recirculation whose center is closer to the moving wall. As the Reynolds number is increased, this center is accompanied with small counter-recirculating vortices at the stagnant corners at the bottom of the cavity as shown for example by Bozeman and Dalton [10]. Some workers have further reported yet smaller recirculating vortices at the stagnant corners, a fact which is supported by similarity solutions of Moffatt [11] and experimental observations (see Van Dyke [12]). Recently, Calvin et al. [13] confirmed these observations by numerical solution of the vorticity-stream function using Newton's method.

To demonstrate the significance of the developments in the present work, we have solved this problem on an extremely fine adapted grid and have shown the existence of quaternary vortices, as in the 10° corner in Figure 2. Table 1 shows that the radius and maximum velocity ratios computed are in close agreement to Moffatt's analysis: $\frac{r_n}{r_{n+1}} = 1.27$, $\frac{v_{n+\frac{1}{2}}}{v_{n+\frac{3}{2}}} = -354.24$.

Vortex Number: n	Radius Ratio: $\frac{r_n}{r_{n+1}}$	Maximum Velocity Ratio: $\frac{v_{n+\frac{1}{2}}}{v_{n+\frac{3}{2}}}$
1	1.26	-322.7
2	1.27	-338.8
3	1.27	-361.0
4	1.29	-560.5

Table 1
Flow in 10° corner

Performance measurements were made with a square cavity at $Re = 100$ on an Intel iPSC/860 machine, and are summarized in Table 2. The calculation was started with a uniform grid of 4096 triangular cells. The grid was adapted near both corners at the bottom of the cavity to capture secondary and tertiary recirculations. The final adapted grid contained 18094 cells. Six adaption cycles were performed with 500 iterations per cycle. The grid was adapted to spatial regions in the vicinity of the stagnant corners at



(a) Primary Vortex, r=0.953

(b) Secondary Vortex, r=0.753

(c) Tertiary Vortex, r=0.591

(d) Quaternary Vortex, r=0.464

Figure 2. Vortices in triangular driven cavity at Re = 0.436

the end of every adaption cycle. After each adaption, the grid was load balanced using the global repartitioning strategy described earlier.

The columns in the table are the number of processors. The first row indicates the overall wall clock, or elapsed, time for six complete adaption cycles. The detailed breakup of the elapsed time includes iteration, adaption and load balance times. Speedups and parallel efficiencies are presented for both elapsed time and iteration time. All calculations presented in the table were load balanced with the exception of the last column (8*). Without load balancing, approximately 40% more time was required using eight processors. The elapsed time speedups are close to those calculated from iteration times, indicating that for problems such as this, where several iterations are performed between adaptions, reasonable parallel efficiencies are achievable.

REFERENCES

1. A. Jameson, W. Schmidt, and E. Turkel. Numerical solution of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes. AIAA 91-1259,

	1	2	4	8	8*
Elapsed Time (s)	31212.3	16417.3	8804.4	5374.6	7411.7
Iteration Time (s)	31075.9	16266.4	8565.5	5077.5	7239.2
Adaption Time (s)	136.4	129.8	178.1	218.8	172.4
Adaption Overhead (%)	.44	.79	2.02	4.07	2.33
Load Balance Time (s)	0	21.1	60.8	78.3	0
Load Balance Overhead (%)	0	.13	.69	1.46	0
Elapsed Time Speedup	1	1.90	3.54	5.80	4.21
Elapsed Time Efficiency (%)	100	95.1	88.6	72.6	52.6
Iteration Time Speedup	1	1.91	3.63	6.12	4.29
Iteration Time Efficiency (%)	100	95.5	90.70	76.5	53.7

Table 2
Performance of parallel algorithm

June 1981.

2. Roe, P. L. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, **43**:357–372, 1981.
3. T. J. Barth and D. C. Jespersen. The design and application of upwind schemes on unstructured meshes. AIAA 89-0366, January 1989.
4. W. Smith. Multigrid solution of transonic flows on unstructured grids. Presented at the ASME Winter Annual Meeting, Dallas, Texas, November 1990.
5. J. M. Weiss and W. A. Smith. Solution of unsteady, low Mach number flow using a preconditioned, multi-stage scheme on an unstructured mesh. AIAA Paper 93-3392, 11th AIAA CFD Conference, Orlando, Florida, July 1993.
6. J. M. Weiss and W. A. Smith. Preconditioning applied to variable and constant density time-accurate flows on unstructured meshes. AIAA Paper 94-2209, 25th AIAA Fluid Dynamics Conference, Colorado Springs, Colorado, June 20-23, 1994.
7. Tysinger, T. L. and Smith, W. A. "An Unstructured Multigrid Algorithm for Parallel Simulation of Fluid Flow". In A. Ecer, J. Periaux and N. Satofuka, editor, *Proceedings of Parallel CFD'94*, Kyoto, Japan, May 1994.
8. Spragle, G.S., Smith, W.A., and Yadlin, Y. "Application of an Unstructured Flow Solver to Planes, Trains and Automobiles". AIAA Paper 93-0889, 31th Aerospace Sciences Meeting, Reno, Nevada, January 11-14, 1993.
9. R. Löhner, R. Ramamurti, and D. Martin. A parallelizable load balancing algorithm. AIAA-93-0061, 1993.
10. J.D. Bozeman and C. Dalton. Numerical study of viscous flow in a cavity. *Journal of Computational Physics*, 12:348–363, 1973.
11. H.K. Moffatt. Viscous and resistive eddies near a sharp corner. *Journal of Fluid Mechanics*, 18:1–18, 1964.
12. M. Van Dyke. *An Album of Fluid Motion.* The Parabolic Press, Stanford, California, 1982.
13. C.J. Ribbens, L.T. Watson, and C.Y. Wang. Steady viscous flow in a triangular cavity. *Journal of Computational Physics*, 112:173–181, 1994.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Distributed CFD on Cluster of Workstations involving Parallel Unstructured Mesh Adaption, Finite-Volume-Galerkin Approach and Finite-Elements

P. Le Tallec, B. Mohammadi, Th. Sabourin, E. Saltel
INRIA
Domaine de Voluceau, 78153 Le Chenay, France.

1. Introduction

Mesh generation and partitioning is a crucial bottleneck in all complex calculations. Generating an unstructured adapted mesh of more than one million of points and partitioning it in equilibrated compact subdomains is still a big challenge.

Our idea to overcome this bottleneck is to include the mesh generation step within the parallel calculation loop. The global strategy is organised as follows:

- generation of a global coarse mesh describing the flow geometry,
- automatic mesh partitioning at coarse level,
- adaptive correcting loop :
  1. parallel adaptive unstructured subdomain remeshing,
  2. subdomain by subdomain parallel solution of turbulent Navier-Stokes equations with explicit interface coupling.

In this paper, we present a two-dimensional parallel adaptive mesh generator integrated in the above loop. In this process, the specific points to be adressed are first the definition and meshing of proper interfaces between the subdomains and second the definition of adequate input and output data structures for each subdomain mesh generator. With these new tools, each subdomain is then treated in parallel, that is we use in parallel the same automatic adaptive mesh generator on each subdomain, and then solve separately on each domain. The adaptive mesh generator to be used locally is the present INRIA Delaunay Voronoi mesh generator which equidistributes mesh points using a residual based local metric. We use a metric based on the second derivative of the pressure for Euler computations and of the entropy for viscous computations but more general metric definition is available in the contexte of systems of PDE [1].

2. Theoretical and Numerical Framework

Our model problem is the compressible Navier-Stokes equations :

$$\frac{\partial W}{\partial t} + \nabla . F(W) = 0 \text{ on } \Omega,$$

with suitable boundary conditions. Here, the unknown W correspond to the conservation variables:

$$W = \begin{bmatrix} \rho \\ \rho u \\ \rho(e + \frac{u^2}{2}) \end{bmatrix} = \text{ conservation variables}$$

and the total flux F is splitted into a convective and a diffusive part as

$$F(W) = F_{conv} + F_{diff} = \text{ total flux}$$

with

$$F_{conv} = \begin{bmatrix} \rho u \\ \rho u \otimes u + pId \\ (\rho(e + \frac{u^2}{2}) + p)u \end{bmatrix}, \; F_{diff} = \begin{bmatrix} 0 \\ -\sigma_v(\mu, \nabla u) \\ -\sigma_v \cdot u + q \end{bmatrix}$$

These equations are discretized by a Finite volume - Finite element scheme. First, the domain Ω of the flow is discretized using node centered cells C_i for the convective part and P_1 finite elements for diffusion, both being defined on a common unstructured grid.

After discretisation, the equations at time step n and node i, are given by

$$\int_{C_i} \frac{W_i^{n+1} - W_i^n}{\Delta t} + \nabla F(W^{n+1})_i = 0$$

$$\nabla F(W^{n+1})_i = \sum_{j \in V(i)} \int_{\partial C_i \cap \partial C_j} F_{conv}(W^{n+1})_{ij} \cdot n_i + \int_{\Omega} F_{diff}(W^{n+1}) \cdot \nabla(\hat{W}_i) + \int_{\partial C_i \cap \partial \Omega} F(W^{n+1}) \cdot n_i.$$

Above, the convective flux $F_{conv}(W^{n+1})_{ij} \cdot n_i$ on each interface is computed

- either by the FDS Roe or Osher or by the FVS kinetic or hybrid strategies, with second order MUSCL approximation of interface variables (this approach will be denoted **(FVG)**),
- or treated as the diffusive flux with the usual local modification of shape function $\hat{W}_i$ (this approach will be denoted **(SUPG)**).

The resulting algebraic system can be solved by an explicit Runge-Kutta scheme:
Loop on n until steady state ($n \approx 1000 - 10\,000$), then loop on Runge-Kutta substeps p ($p = 1, 2$ or 3)

$$\int_{C_i} \frac{W_i^{n+1,p} - W_i^n}{\alpha_p \Delta t_i} + \nabla F(W^{n+1,p-1})_i = 0.$$

This algorithm is implemented as a succession of different loops, which are called at each time step n:

1. Loop on Elements :
   - gather nodal values of W,
   - compute diffusive fluxes $\int_E F_{diff}(W) \cdot \nabla(\hat{W}_i)$ and gradients ∇W (for MUSCL reconstruction),
   - scatter (send and add) results to nodal residuals and gradients.
2. Loop on Edges (not for **SUPG**) :
   - gather nodal values of W and ∇W,
   - compute convective fluxes $\int_{\partial C_i \cap \partial C_j} F_{conv}(W^{n+1})_{ij} \cdot n_i$,
   - scatter results to nodal residuals.
3. Loop on Nodes : update nodal values as predicted by algebraic solver

$$W_i^{n+1,p} = W_i^n - \alpha_p \Delta t_i \nabla F(W^{n+1,p-1})_i.$$

The fluid solver used here is called NSC2KE and is in free access at *Bijan.Mohammadi@inria.fr.*

3. Parallel Implementation

The parallel implementation is achieved by *PARALLEL REGIONS*. In other words, the physical domain is first shared into nonoverlapping subdomains. The same CFD code is then run in parallel on each region. Compared to a monodomain approach, this requires the addition of two new tasks:

1. automatic mesh partitioning into subdomains having small interfaces and a balanced number of elements,

2. explicit communication step between regions for sending and adding information to interface nodes of neigboring subdomains at each scattering step (one or two communications per time step).

This strategy is very attractive, and has been tested and implemented by different teams for two and three dimensional configurations [2], [3].

3.1. Parallel mesh generation and solver

The integration of local parallel remeshing capabilities in a standard parallel Navier-Stokes solver can be done within the new algorithm structure that we propose below:

A) Sequential Initialisation

- initial calculation of the flow on a coarse mesh,
- construction of a local metric (absolute value of inverse Hessian of pressure or entropy field) to weight the regions of interest,
- if desired, coarse remeshing of the flow domain equireparting elements areas following the available local metric,
- automatic partition of the coarse mesh into a given number of compact subdomains with same weighted number of elements (weight = initial/final volume),
- geometric definition of the interfaces in this partition. The corresponding subdomains will then be defined by their list of interfaces.

B) Parallel correction loop (one processor per subdomain)

- parallel adaptive fine meshing of the interfaces on a central processor,
- send interfaces to relevant subdomain processors,
- local fine adaptive mesh of each subdomain (equidistributed for the given local metric),
- independent subdomain solve of Navier-Stokes (with synchronous scatter at each time step),
- local error estimates and calculation of a new local metric.

The realisation and implementation of such a parallel loop requires additional specific tools which are listed below.

1. parallel programming environment : PVM is the standard choice and is the one which was used in the examples presented below. Friendly new versions of such communication libraries will be even better.

2. mesh partitioner : when dealing with large, complex structures, the partitioning of the original structure in many subdomains cannot be done by hand. Automatic tools are then needed which must achieve three goals :

   - the number of interface nodes must be as small as possible in order to reduce the size of the interface problem or the number of communications and synchronizations,

- the amount of work necessary for solving each subdomain should be well-balanced in order for them to be solved efficiently in parallel on different processors,

- the shape of the different subdomains should be regular in order to have extensions maps with small energies when using global implicit solvers.

Different partitioners are now avalaible in the research community . The simplest is probably the greedy algorithm. In this strategy, all subdomains are built by a frontal technique. The front is initialized by taking the most isolated boundary node, and is advanced by adding recursively all neigboring nodes until the subdomain preassigned size has been reached. Another popular technique is the recursive dissection algorithm in which the original mesh is recursively cut into halves. In the simplest case, at each dissection step, the new interface is perpendicular to the maximal diameter of the considered domain, or to one of its principal axis of inertia. In a more elaborate and expensive strategy, the interface is taken as the surface of zero values of the second eigenvector of a Laplace operator defined on the global mesh. It can be shown that the corresponding solution approximately minimizes the interface size, and leads to rather regular shapes. We have used a more recent approach, based on the "K means" techniques used in automatic data classification. This algorithm, iteratively improves preexisting partitions by first computing the barycenters of the different subdomains, and by affecting then any element to the subdomain with the closest barycenter (measured by an adequate weighted norm).

3. Interface Data Structure : a specific data structure has to be created for the description of the different interfaces. Each interface is defined independently at two levels. The first level refers to its geometric definition. In our implementation, it is characterised by an ordered list of vertices and normals. This list is unchanged during the mesh adaption process and is the result of the automatic partitioning of a global coarse mesh of the domain under study. Between these nodes, the interface is defined by piecewise Hermite cubic interpolation. The second level defines the current discretisation of the interface, and is constantly updated during the adaption process. Body surfaces or artificial internal boundaries are treated by the same technique. The accuracy of the generated interface geometry is simply controlled by the number of points used at the geometric level. Additional pointers enable an easy exchange of information between subdomains and interfaces. As a consequence, information exchange between neighboring subdomains will be achieved by sharing information on their common interface. The selected data stuctures have three main advantages: they are perfectly local, they can handle any type of geometry and partitioning structure, and the adaption criteria is summarized in a unique metric tensor defined locally at each node.

4. Interface and subdomain mesh generators : we need here to adapt the existing mesh generators in order for them to be compatible with the selected data structure and to handle general local metrics. This point is described in detail in the next section.

5. Better error estimation and calculation of local metrics are now available [1].

Remark 1 *The PVM implementation requires message passing between neighboring subdomains during the solution phase. Moreover, it activates at each time step a decision maker processor. This processor receives the local residuals from each subdomain. It then computes the global residual and decides either on the continuation of the solution loop, or on the execution of a remeshing step, or on the end of the global job.*

When remeshing is decided, each processor receives the order of computing its local metric. Interface metrics are then collected on a single interface processor which uses them to mesh interfaces. Afterwards, each processor proceeds to generate its internal subdomain mesh and restarts the execution loop.

4. Two-dimensional Adaptive Remeshing

We detail in this section the adaptive control strategy that we have used locally for generating interface or subdomain meshes. We have used a Delaunay-Voronoi unstructured mesh generator and anisotropic control variables to specify both the sides and the stretching of the generated elements.

Standard isotropic control strategies operating on bounded connected domains Ω use a local refinement measure r defined on Ω with values in R^+ which determines at each position the desired size of the

generated triangle. The mesh generator then attempts to generate equilateral triangles whose side will be of length $r(x)$ if the triangle is centered at x.

Anisotropic control strategies are usually more efficient and flexible [4]. For such strategies, each node is associated to a metric tensor ($\lambda_1 > 0, \lambda_2 > 0, \alpha$) in which to measure the characteristic length of the mesh. An adapted mesh is then a uniform mesh with respect to the metric locally defined by these tensors. To be more specific, let us first recall that an euclidian metric on a vector space X is a positive mapping m measuring a distance $m(x,y)$ between two points x and y of X and such that $\forall x, y, z \in X$:

$$m(x,y) = m(y,x)$$

$$m(x,y) \leq m(x,z) + m(x,z)$$

$$m(x,y) = 0 \Longleftrightarrow x = y.$$

Among all possible metrics, we will only consider those generated by an underlying scalar product $m(x,y) =< x-y, x-y >^{1/2}$ where

$$< x, y >= x^t M y.$$

Here, $M(u)$ is a symmetric definite matrix depending on the position u in Ω. Any given stretched triangle can actually be associated to such a scalar product. More precisely, for any triangle K, there exists a unique scalar product M in which K is equilateral. Therefore, defining the size, shape and orientation of the local triangles is equivalent to the definition of M. In R^2, the matrix M is given by its spectral decomposition

$$M = \begin{pmatrix} a & b \\ b & c \end{pmatrix} = R(\alpha) \begin{pmatrix} \lambda_1 & 0. \\ 0. & \lambda_2 \end{pmatrix} R^{-1}(\alpha)$$

where $\lambda_1 > 0$ and $\lambda_2 > 0$ are its eigenvalues and where

$$R(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$$

is a rotation matrix of angle α. Unit equilateral triangles in this metric are of length $1/\sqrt{\lambda_1}$ in the direction α and $1/\sqrt{\lambda_2}$ in the orthogonal direction.

The local values of $\lambda_1 > 0$, $\lambda_2 > 0$ and α are the final control variables characterizing locally our ideal adapted mesh. In this framework, the adaption strategy simply consists in constructing these local values. Our choice then attempts to realize an equirepartition of the interpolation error of a given field u

$$\| u - \pi_h u \|_{L^2(K)} \approx c \quad h_K^2 |u|_{H^2(K)}$$

along each given direction. Therefore, the spacing h_l along direction e_l should satisfy

$$h(x)_l^2 |D_l^2(\pi_h u)| = c \quad \forall l.$$

Working along the principal directions x_1 and x_2 of the Hessian matrix of u, and denoting by λ_1 and λ_2 the absolute values of the associated eigenvalues, we deduce :

$$h_1^2 |\lambda_1| = h_2^2 |\lambda_2| = c.$$

The final metric is then characterized by h_1, h_2 and the angle α the first eigenvector of this Hessian matrix. In practice, the Hessian matrix is obtained from the pointwise knowledge of the field u through the formula

$$\frac{\partial^2 \pi_h u}{\partial x_i \partial x_j}(S_k) = \frac{-\int_\Omega \frac{\partial \pi_h u}{\partial x_j} \frac{\partial \varphi}{\partial x_j}}{\int_\Omega \varphi}$$

with φ the finite element shape function with value 1 at node S_k and 0 everywhere else. The only remaining choice is then to specify the physical field u used in such a construction. Pressure or entropy are good candidates.

Once this metric is defined, each interface is first discretised. This is done looping on the different interfaces. For each interface, we equidistribute the boundary nodes on the interface following the local metric defined at each vertex. The spacing of two successive nodes for this metric is taken to be constant. The intermediate nodes which are added are projected on the piecewise cubic defined by Hermite or linear interpolation on the vertices given at the geometric level.

Then, we proceed to the local independent fine adaptive mesh generation of the different subdomains. The program iteratively generates internal nodes in addition to the boundary nodes previously created along the interfaces. These new nodes are equidistributed with respect to the local metric and are connected in a triangular mesh by using a Delaunay type algorithm.

5. Numerical Results

We are particularly interested by testing the ability of our algorithm in managing a correct interface resolution between subdomains as no overlaping has been operated and that the points on interfaces move.

We present results for a supersonic Euler flow over a bi-naca and a Navier-Stokes computation over a cylinder at Mach 0.4 and Reynolds 80. Four subdomains have been used for these cases. The non-overlaping strategy works quite well and introduces no perturbation in the solution. We can see that the load of the different subdomains are completely unballanced at the end of the adaption loop.

6. Conclusion

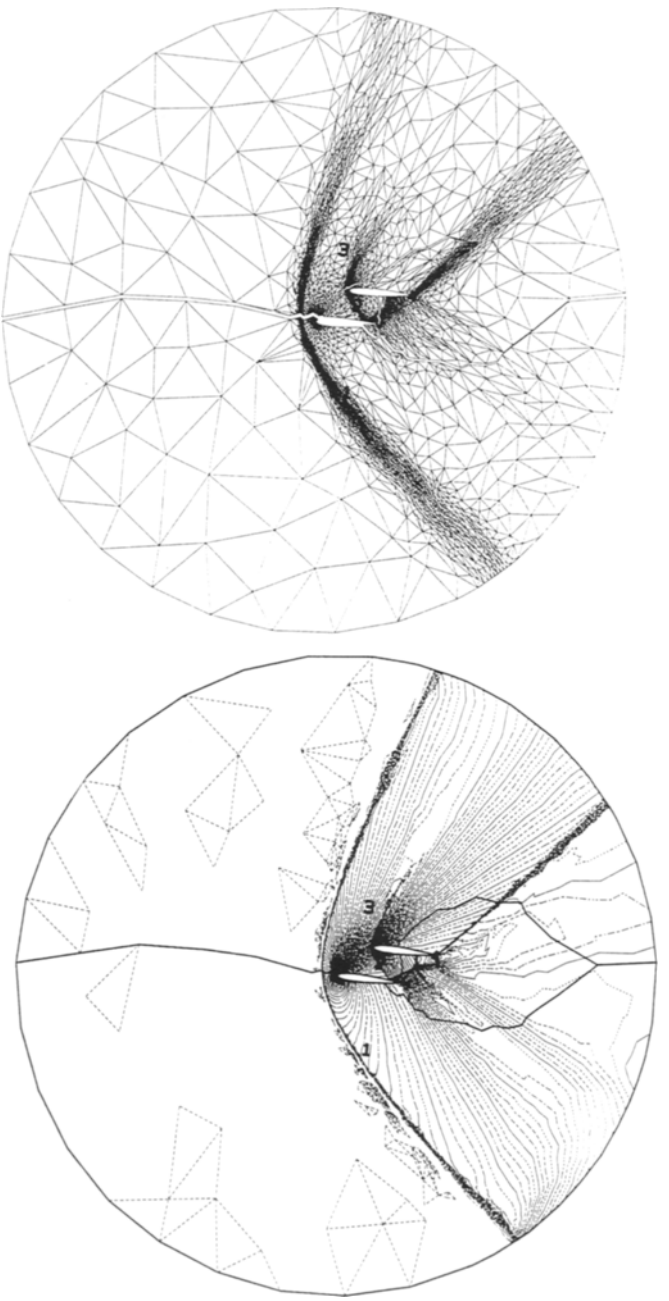
We have presented here the results of our adaptive parallel mesh generator and the integration of this mesh generator within a parallel solution of the Navier Stokes equations.

The present strategy does not reequilibrate the different subdomains after remeshing. Therefore, it is important to generate from the beginning a partition which stays balanced after remeshing. This can only be achieved if the partitioner knows the result of a coarse initial calculation from which it can derive an approximation of the final metrics to be locally used inside each subdomain. Otherwise, dynamic balancing must be implemented, but this is by large an open problem. So, at this time no particular care has been taken to correct a bad load balancing and this is probably the major weakness of this technique.

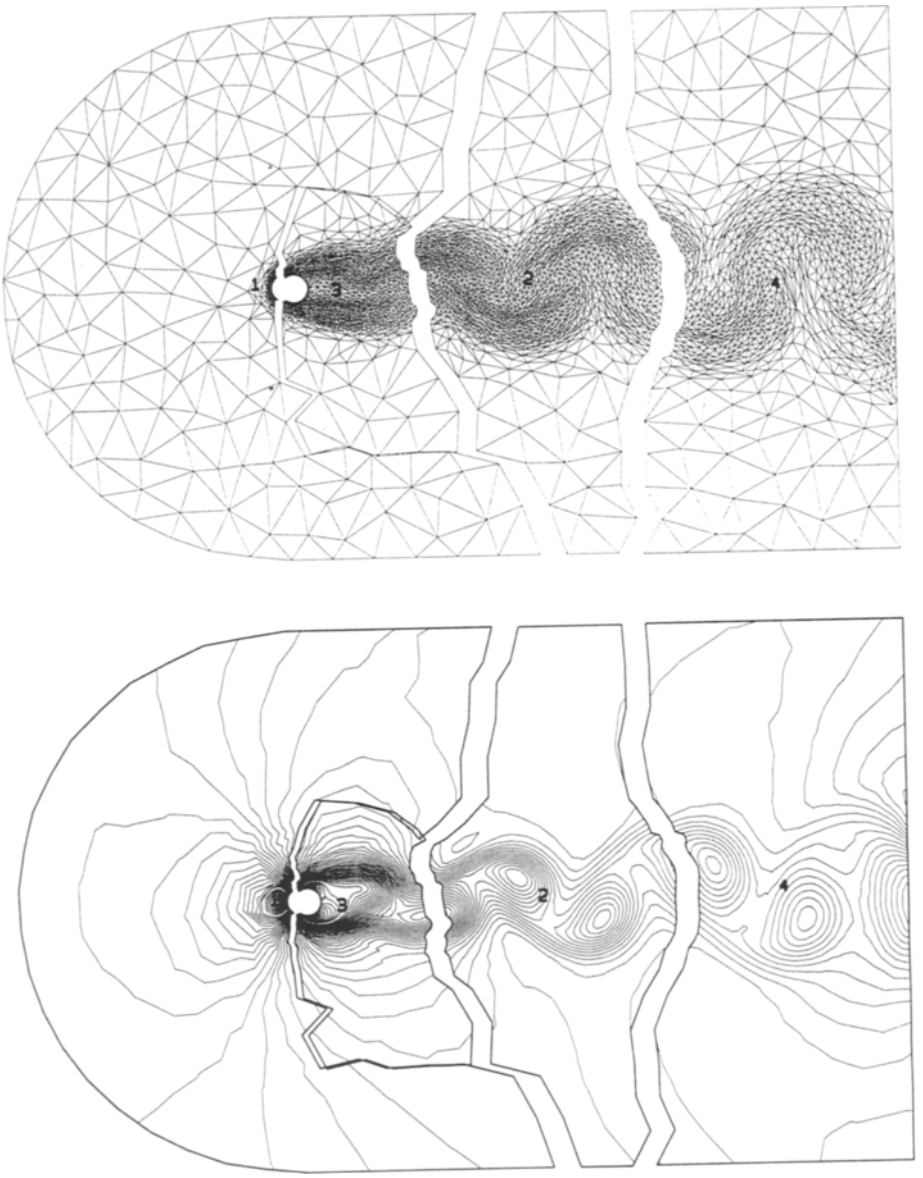
REFERENCES

1. M. Castro, F. Hecht, B. Mohammadi, *New Progress in Anisotropic Mesh Generation, Application to Viscous and Inviside Flows*, Proceedings of the 4th International Meshing Roundtable, 1995.
2. S. Lanteri, *Simulation d'écoulements Aérodynamiques Instationnaires sur une architecture S.I.M.D. Massivement parallèle* , Thèse de l'université de Nice Sophia-Antipolis, (1991).
3. Th. Sabourin,*Simulation d'Ecoulements 3D sur supercalculateurs parallèles*, Thèse d'Université Paris Dauphine, 1994.
4. M.G. Vallet, *Génération de Maillages Eléments Finis Anisotropes et Adaptatifs*, Thèse de doctorat de l'université de Paris VI (1992).

Binaca: Euler computation at Mach 1.1 on four subdomains
Final mesh and iso-density lines



Flow over cylinder: Subsonic Navier-Stokes computation
at Mach 0.4 and Reynolds 80 on four subdomains
Final mesh and iso-density lines



Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

Semi-Lagrangian Shallow Water Modeling on the CM-5

B. T. Nadiga[a], L. G. Margolin[a], and P. K. Smolarkiewicz[b]

[a]Los Alamos National Lab., MS-B258, Los Alamos, NM 87545

[b]National Center for Atmosperic Research, Boulder, CO 80307

We discuss the parallel implementation of a semi-Lagrangian shallow-water model on the massively parallel Connection Machine CM-5. The four important issues we address in this article are (i) two alternative formulations of the elliptic problem and their relative efficiencies, (ii) the performance of two successive orders of a generalized conjugate residual elliptic solver, (iii) the time spent in unstructured communication—an unavoidable feature of semi-Lagrangian schemes, and (iv) the scalability of the algorithm.

1. The Physical Problem: A SubTropical-SubPolar Gyre System

We model the wind-driven circulation in a closed rectangular midlatitude ocean basin. The model consists of a dynamically active shallow surface layer overlying an infinitely deep lower layer and the chosen wind forcing, acting on the surface layer, results in the formation of a double gyre. The vertically-averaged circulation of the surface layer can then be modelled using the hydrostatic shallow water equations [1]:

$$\frac{\partial \eta}{\partial t} + \boldsymbol{\nabla}\cdot(\eta \mathbf{u}) = 0, \tag{1a}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\boldsymbol{\nabla}\mathbf{u} = -g\frac{\Delta\rho}{\rho}\boldsymbol{\nabla}\eta + f\hat{\mathbf{z}}\times\mathbf{u} + \sigma_x(y)\hat{\mathbf{x}}. \tag{1b}$$

where η is the fluid thickness and $\mathbf{u}$ is the horizontal velocity vector. The domain size is 1000 kms in the east-west direction and 2000 kms in the north-south direction; the initial undisturbed depth of the upper layer is 500 meters. The reduced gravity parameter, $g\Delta\rho/\rho(\equiv g')$, where $\Delta\rho$ is the difference in densities between the upper and lower layers, is 0.031, the wind stress profile $\sigma_x(y) = -A\cos(2\pi y/D)$, where A corresponds to a wind stress amplitude of 0.1 N/m^2. The simulations are carried out on a beta plane, *i.e.*, $f = f_0 + \beta y$, with $f_0 = 5\ 10^{-5}$ s^{-1} and $\beta = 2\ 10^{-11}$ m^{-1}s^{-1}. The model ocean is started from rest and the performance comparisons of the various implementations are done during the spin up period consisting of the first 30 days (at the end of which all implementations give identical results).

2. The Semi-Lagrangian Scheme

To better illustrate the issues encountered in our parallel implementation, we briefly describe the semi-Lagrangian technique. Equations 1 have two different types of terms: advective or streaming terms, and the remaining terms which represent forces acting on Lagrangian fluid particles. To better treat the different natures of the two types of terms, the semi-Lagrangian technique employs separate discretizations for them—integrating the forces along a parcel trajectory (the Lagrangian aspect) and evaluating the advection terms along a residual contour [2]. This differential treatment is in contrast to the single

form of discretization for any term, regardless of its physical nature, employed in an Eulerian scheme.

Equations 1 may be rewritten symbolically in the form

$$\frac{d\psi}{dt} = \mathbf{R}, \tag{2}$$

where $\psi = \psi(\mathbf{x}, t)$ is a fluid variable (η or $\mathbf{u}$), $\mathbf{R} = \mathbf{R}(\mathbf{x}, t)$ is the corresponding force expressible in terms of ψ and its derivatives, and $d/dt \equiv \partial/\partial t + \mathbf{u}\cdot\nabla$ is the material time derivative. Integration of (2) in time leads to

$$\psi(\mathbf{x}, t) = \psi(\mathbf{x}_o, t_0) + \int_T \mathbf{R}\, dt, \tag{3}$$

where T is the parcel trajectory connecting the arrival point $(\mathbf{x}, t)$, assumed to coincide with a grid point, to $(\mathbf{x}_o, t_0)$, the corresponding departure point, not necessarily a grid point. We compute the departure point $\mathbf{x}_o$ corresponding to the arrival point $\mathbf{x}$ to second-order accuracy by two iterations of

$$\mathbf{x}_o = \mathbf{x} + \int_{t_1}^{t_0} \mathbf{u}(\mathbf{x}, t)\, dt. \tag{4}$$

Since the departure point will not in general coincide with a grid point, an interpolation procedure has to be invoked in order to determine the discrete field value $\psi(\mathbf{x}_o, t_0)$ in (3). This interpolation is equivalent to solving a constant coefficient advection problem locally at the departure point, with the constant coefficient being proportional to the separation of the departure point $\mathbf{x}_o$ from its nearest grid point $\mathbf{x}_*$ [2]. Thus (3) may be rewritten as

$$\psi(\mathbf{x}, t) + \mathrm{O}(\mathcal{A}) = \mathcal{A}\,(\psi(\mathbf{x}_*, t_0), \Upsilon_0) + \int_T \mathbf{R}\, dt, \tag{5}$$

where $\mathcal{A}$ is a constant coefficient Eulerian advection operator, $\mathrm{O}(\mathcal{A})$ is the truncation error associated with $\mathcal{A}$, and $\Upsilon = (\mathbf{x}_* - \mathbf{x}_o)/\Delta\mathbf{x}$ is the nondimensional separation of the departure point $\mathbf{x}_o$ from its nearest grid point $\mathbf{x}_*$, and represents the Courant number for the advection problem. In view of the constancy of the advective velocity, the multi-dimensional advection operator can be replaced *exactly* by a combination of one-dimensional advection operators (*i.e.*, there are no splitting errors) and $\mathcal{A}$ can be chosen to be a one-dimensional constant coefficient Eulerian advection operator. Thus, the entire semi-Lagrangian flow model can be built upon a one-dimensional Eulerian advection scheme, while retaining the formal accuracy of its constant-coefficient limit. We use the second-order flux-corrected Lax-Wendroff scheme for $\mathcal{A}$ (*e.g.*, see [2] and references therein).

Semi-Lagrangian techniques are usually found to be more accurate than Eulerian techniques, but are not formulated in terms of a detailed balance of fluxes and so are not exactly conservative. Another recognized advantage of the semi-Lagrangian technique is the relaxation of the Courant-Friedrichs-Lewy (CFL) stability criterion (based on the velocity) to a less restrictive condition based on the spatial derivatives of velocity.

3. Semi-Lagrangian Discretization of the Shallow Water Equations

After adding provision to absorb waves at the boundaries without reflection (restoring boundaries), (1a) may be rewritten as

$$\frac{d\eta}{dt} = -\eta\nabla\cdot\mathbf{u} - \frac{\eta - \eta_a}{\tau(\mathbf{x})}. \tag{6}$$

A discretization of (6) centered on the midpoint of T leads to

$$\frac{\eta-\eta_o}{\Delta t} = -\frac{1}{2}\{\eta_o \nabla\cdot\mathbf{u} + \eta[\nabla\cdot\mathbf{u}]_o\} - \frac{1}{2}\left\{\left[\frac{\eta-\eta_a}{\tau}\right]_o + \frac{\eta-\eta_a}{\tau}\right\}. \tag{7}$$

where we have used harmonic averaging in the midpoint evaluation of the first term on the right hand side of (6) to linearize the equation in the as yet unknown values of the fluid variables at the arrival point. ψ_o is a short-hand notation for $\psi(\mathbf{x}_o, t_0)$, and is estimated using the second-order procedure $\mathcal{A}(\psi(\mathbf{x}_*, t_0), \Upsilon_0)$ of (5). The momentum equation is, again with provision for restoring boundaries,

$$\frac{d\mathbf{u}}{dt} = -g'\nabla\eta + f\hat{\mathbf{z}}\times\mathbf{u} + \sigma_x(y)\hat{\mathbf{x}} - \frac{\mathbf{u}-\mathbf{u}_a}{\tau}, \tag{8}$$

and a similar discretization of it leads to

$$\begin{aligned}\frac{\mathbf{u}-\mathbf{u}_o}{\Delta t} = &-\frac{1}{2}g'(\nabla\eta + [\nabla\eta]_o) + \frac{1}{2}(f\hat{\mathbf{z}}\times\mathbf{u} + [f\hat{\mathbf{z}}\times\mathbf{u}]_o)\\ &+\frac{1}{2}(\sigma_x(y) + [\sigma_x(y)]_o)\hat{\mathbf{x}} - \frac{1}{2}\left(\frac{\mathbf{u}-\mathbf{u}_a}{\tau} + \left[\frac{\mathbf{u}-\mathbf{u}_a}{\tau}\right]_o\right).\end{aligned} \tag{9}$$

Equations (7) and (9) (with centered differencing for ∇), form the basis of our spatially and temporally second-order accurate, two-time level, semi-Lagrangian model of (1).

3a. Implicit Free Surface: Velocity Formulation

Equation (7) can be solved explicitly for the depth field at the arrival point η in terms of the divergence of the velocity field as

$$\eta = \alpha - \beta\nabla\cdot\mathbf{u}, \tag{10}$$

$$\alpha = \left\{\left[\eta - \frac{\Delta t}{2\tau}(\eta-\eta_a)\right]_o + \frac{\Delta t}{2\tau}\eta_a\right\}\{1+\Delta t/2([\nabla\cdot\mathbf{u}]_o + 1/\tau)\}^{-1},$$

$$\beta = \frac{\Delta t}{2}\eta_o\{1+\Delta t/2([\nabla\cdot\mathbf{u}]_o + 1/\tau)\}^{-1}.$$

Further, (9) can be simplified to express $\mathbf{u}$ in terms of $\nabla\eta$ as follows:

$$\mathbf{A}\mathbf{u} = \tilde{\mathbf{u}} - \frac{\Delta t}{2}g'\nabla\eta, \tag{11}$$

$$\mathbf{A} = \begin{pmatrix} 1+\frac{\Delta t}{2\tau} & -f\frac{\Delta t}{2} \\ f\frac{\Delta t}{2} & 1+\frac{\Delta t}{2\tau}\end{pmatrix}, \qquad \tilde{\mathbf{u}} = \left[\mathbf{u} + \frac{\Delta t}{2}\left(-g'\nabla\eta + f\hat{\mathbf{z}}\times\mathbf{u} + \sigma_x(y)\hat{\mathbf{x}} - \frac{\mathbf{u}-\mathbf{u}_a}{\tau}\right)\right]_o + \frac{\Delta t}{2}\left(\sigma_x(y)\hat{\mathbf{x}} + \frac{\mathbf{u}_a}{\tau}\right).$$

We insert (10) into (11) to obtain a variable-coefficient linear elliptic equation for $\mathbf{u}$:

$$\mathcal{L}(\mathbf{u}) = \frac{\Delta t}{2}g'\nabla(\beta\nabla\cdot\mathbf{u}) - \mathbf{A}\mathbf{u} = \frac{\Delta t}{2}g'\nabla\alpha - \tilde{\mathbf{u}}, \tag{12}$$

where the right-hand-side and the coefficients $\mathbf{A}$ and β are all known.

Equations (12) and (10) constitute our implicit free surface semi-Lagrangian shallow water model formulated in terms of an elliptic equation for the velocity field. This formulation is particularly useful when dispersion is introduced at the next higher level of approximation (*e.g.*, the Green-Naghdi equations [3]) of the shallow fluid equations. Here the introduction of singular dispersive terms in the velocity equations necessitates an implicit treatment of those terms in order to allow an affordably large time step. We will discuss these higher-order formulations elsewhere.

3b. Implicit Free Surface: Depth Formulation

Alternately, (11) may be solved analytically for the velocity $\mathbf{u}$ in terms of $\nabla\eta$:

$$\mathbf{u} = \mathbf{A}^{-1}\left(\tilde{\mathbf{u}} - \frac{\Delta t}{2} g' \nabla\eta\right). \tag{13}$$

Then, inserting (13) in (10), we again obtain a variable-coefficient linear elliptic equation, but now for η:

$$\mathcal{L}(\eta) = \frac{\Delta t}{2} g' \beta \nabla\cdot\left[\mathbf{A}^{-1}\nabla\eta\right] - \eta = \beta\nabla\cdot\left[\mathbf{A}^{-1}\tilde{\mathbf{u}}\right] - \alpha. \tag{14}$$

Equations (14) and (11) constitute our semi-Lagrangian shallow water model, formulated now in terms of an elliptic equation for the depth field. We note that this formulation does not extend to the next higher level of approximation of the shallow fluid equations since the higher-order dispersive terms that would then appear in the momentum equation do not allow an explicit solution of $\mathbf{u}$ in terms of $\nabla\eta$, as we have done here in (13).

4. Implementation

We have implemented both formulations (Eqs. (12) & (10) and Eqs. (14) & (11)) on the massively parallel Connection Machine CM-5 at Los Alamos National Laboratory. Each node of the CM-5 has four vector units and we use the data parallel mode of computation, programming in CM Fortran. The problem size is fixed at 256 grid points in both the latitude and longitude directions unless otherwise specified and we use partitions of three different sizes—either 32 or 64 or 128 nodes. In all implementations, we use the default array layout with both the latitude and longitude axes declared parallel. (Performance was severely degraded by declaring one of the axes serial.) We base our computations at the arrival point to avoid difficulties of balancing the load. All computations are performed in single precision 32 bit mode. We have verified the sufficiency of single precision calculations by comparisons with double precision computations in steady flow cases. Care was taken in the coding to make the parallel code blocks as long as possible by separating out the statements that required communications from those that did not. We did not use the optimization flag on the compiler.

Table 1 shows where the program spends most of the time. The timings are given for the implementation **UC4** discussed later. All times reported in this article were obtained using the '*-cmprofile*' option of the CM Fortran compiler and analysed using 'prism'. In addition to the total time spent by the procedures on the nodes, we have considered the usage of four different resources on the CM-5 to characterise performance.

1. Node CPU—the time the program spent in processing on any of the nodes.
2. NEWS—the time spent in structured grid communication (*e.g.*, cshift).
3. Send/Get—the time spent in router communication (*e.g.*, indirect addressing).
4. Reduction—the time spent in data reductions (*e.g.*, global sum/maximum/minimum).

Procedure	Node Total	Node CPU	Communication		
			NEWS	Send/Get	Reduction
Total	275.3	169.8	63.7	31.7	10.1
$\mathcal{A}(\psi(\mathbf{x}_*, t_0), \Upsilon_0)$	149.7	100.8	17.2	31.7	0.0
$\mathcal{L}(\psi) = \mathcal{R}$	109.1	56.8	42.2	0.0	10.1

Table 1: The time spent (in seconds) in the advection of various quantities and the solution of the elliptic equation for a 30 day run on 32 nodes.

There are seven calls to the advection routine and one call to the elliptic solver. The program spends about 50% of the time in advection or interpolation of various quantities and about 40% of the time in the solution of the elliptic equation. Below, we consider in detail the performance aspects of the different implementations. Unless otherwise specified, all performance data presented in this article are time spent in seconds on a 32 node partition for a 30 day run using a 256×256 mesh.

4a. Unstructured Communication

The time step in our semi-Lagrangian simulations is determined by restricting $\| \frac{\Delta \mathbf{u}}{\Delta \mathbf{x}} \| \Delta t$ (where the norm used is the L_∞ norm) to be less than 0.5. This condition is much less restrictive than the CFL condition characteristic of Eulerian schemes and allows much larger time steps. The price to be paid for this is unstructured communication or in other words indirect addressing. Owing to architecture of the CM-5, unstructured communication is much slower than the regular shift type of communication. The cost of unstructured communication is reflected in the Send/Get measure. We consider four different ways of implementing the unstructured communication:

	Node Total	Node CPU	Communication		
			NEWS	Send/Get	Reduction
UC1	338.4	168.9	63.1	96.6	9.7
UC2	321.6	162.1	64.1	85.6	9.8
UC3	396.6	162.3	65.1	159.5	9.8
UC4	275.5	169.9	63.6	31.8	10.1

Table 2: Usage of different resources by the four implementations of unstructured communication.

1. **UC1** uses the *forall* construct of CM Fortran and is the closest to the usual Fortran77.
2. **UC2** computes a trace for the unstructured communication pattern once the departure point calculations are done and then repeatedly uses the trace until the departure point changes.
3. **UC3** is similar to **UC2**, but differs by computing the trace in a manner such that all get operations are performed using send operations.
4. **UC4** computes a trace as in **UC2** and in addition, whenever unstructured communication is to be done, packs four single precision real fields into one double precision

complex field. This packing (and consequent unpacking) requires no copying or movement of data, and is achieved by maintaining two different front end array descriptors describing the same parallel memory area.

Methods **UC2**, **UC3**, and **UC4** make use of CMSSL communication routines. In the model implementations using the four methods above, we formulate the elliptic equation in terms of velocity and solve the elliptic equation using **GCR(2)** (see later).

The only significant difference in the performance data for these four cases is in the time spent in unstructured communication. Among the first three implementations, **UC2** takes the least time and hence it was the method of choice for **UC4**. (We note that the method of packing of **UC4** could have been used in conjunction with any of the first three methods.) The reduction in the unstructured communication time in **UC4** is related to the nature of the semi-Lagrangian scheme. As indicated in section 2, the heart of a semi-Lagrangian scheme is a 1D Eulerian advection operator. For our 2D problem, this results in three 1D advections in one direction followed by a combining 1D advection in the other direction, With a nine-point stencil and arrival point based computation, there are thus three sets of data movement, with each set of movement consisting of three field values. In **UC4**, the movement of three field values has been compressed into one movement by packing three single-precision reals into one double-precision complex and this is reflected in a (close to a) factor of three reduction in the time spent in unstructured communication between **UC2** and **UC4**. (In fact four single precision reals can be compressed into one double precision complex, and so 25% of the data movement is wasted.) Since in **UC4**, unstructured communication is no more a bottleneck and takes only about 50% of the time spent in grid communications, we consider the **UC4** method of unstructured communication satisfactory and only use that in all future comparisons. We note that if the computations were to be done in double precision, there would only be a 33% reduction in the Send/Get time in using **UC4** instead of **UC2**. Thus, by packing shorter data types into longer ones and computing a trace which gets used repeatedly till the pattern of communication changes, we effectively minimize time spent in unstructured communications.

4b. Velocity Formulation vs. Depth Formulation

	ϵ	Iters.	Node Total	Elliptic Solver			
				Total	CPU	NEWS	Reduction
VF	10^{-3}	8	228.9	62.5	32.4	24.5	5.7
DF	10^{-3}	5	220.3	51.3	25.0	23.0	3.3
VF	10^{-5}	14	275.5	109.0	56.8	42.9	9.8
DF	10^{-5}	9	255.8	86.8	42.5	38.5	5.9

Table 3: A comparison of the relative efficiencies of formulating the elliptic problem in terms of depth and in terms of the velocity field.

In Table 3, we show the relative performance of the velocity formulation **VF** (Eqs. (12) and (10)) and the depth formulation **DF** (Eqs. (14) and (11)) for two cases—first to reduce the residual in the elliptic equation by a factor of 10^{-3} and next to reduce the residual by a factor of 10^{-5}. It is clear that the depth formulation is consistently more efficient but then, only by about 4-8%. Thus by formulating the elliptic problem in terms of

velocity, while the extra overhead is not large, the formulation readily generalizes to the higher level (dispersive) approximations of the shallow water model. In addition, there is a scope for designing better elliptic solvers with the extra degree of freedom in the velocity formulation.

4c. The Elliptic Solver

We briefly outline the generalized conjugate residual algorithm (*e.g.*, see [4]) we use to solve the elliptic equation $\mathcal{L}(\psi) = \mathcal{R}$. For any initial guess ψ^0, set $\mathbf{p}^0 = \mathbf{r}^0 = \mathcal{L}(\psi^0) - \mathcal{R}$, and then iterate as follows:

For $n = 1$ to convergence do

for $\nu = 0, .., k-1$ do

$$\beta = -\frac{< \mathbf{r}^\nu \mathcal{L}(\mathbf{p}^\nu) >}{< \mathcal{L}(\mathbf{p}^\nu)\mathcal{L}(\mathbf{p}^\nu) >}$$

$$\psi^{\nu+1} = \psi^\nu + \beta \mathbf{p}^\nu; \quad \mathbf{r}^{\nu+1} = \mathbf{r}^\nu + \beta \mathcal{L}(\mathbf{p}^\nu)$$

exit if $\| \mathbf{r}^{\nu+1} \| \leq \varepsilon \| \mathbf{r}^0 \|$

$$\forall_{l=0,\nu} \quad \alpha_l = -\frac{< \mathcal{L}(\mathbf{r}^{\nu+1})\mathcal{L}(\mathbf{p}^l) >}{< \mathcal{L}(\mathbf{p}^l)\mathcal{L}(\mathbf{p}^l) >} \qquad \textbf{GCR(k)}$$

$$\mathbf{p}^{\nu+1} = \mathbf{r}^{\nu+1} + \sum_{l=0}^{\nu} \alpha_l \mathbf{p}^l; \quad \mathcal{L}(\mathbf{p}^{\nu+1}) = \mathcal{L}(\mathbf{r}^{\nu+1}) + \sum_{l=0}^{\nu} \alpha_l \mathcal{L}(\mathbf{p}^l)$$

end do

reset $[\psi, \mathbf{r}, \mathbf{p}, \mathcal{L}(\mathbf{p})]^k$ to $[\psi, \mathbf{r}, \mathbf{p}, \mathcal{L}(\mathbf{p})]^0$

end do

Table 4. shows the relative performances of **GCR(2)** and **GCR(3)**. Though **GCR(3)** performs exceedingly well in terms of the number of iterations to reduce the residual by a factor ϵ compared to **GCR(2)**, there is no speedup either in terms of CPU time or communication time.

	ϵ	Iters.	Node Total	Elliptic Solver			
				Total	CPU	NEWS	Reduction
GCR(2)	10^{-3}	5	220.3	51.3	25.0	23.0	3.3
GCR(3)	10^{-3}	2	225.7	55.5	27.7	23.9	5.7
GCR(2)	10^{-5}	9	255.8	86.8	42.5	38.5	5.9
GCR(3)	10^{-5}	3	265.6	95.2	48.1	39.8	7.3

Table 4: Comparison of the CPU and communication times for **GCR(2)** and **GCR(3)**.

4d. Scalability

We consider the scalability of the algorithm both when the problem size is held constant and the number of processors varied and when the problem size is scaled with the

number of processors. From entries 1 and 4 in Table 5, it is clear that the algorithm and its implementation scale perfectly in the latter case, so that all individual times—the CPU time and the times involved in the different types of communication—remain almost exactly the same.

For the case of scaling when the problem size is held constant, if the subgrid length (*i.e.*, the number of grid points per vector unit) is over 512, the CPU part of the algorithm scales perfectly with number of processors, and the unstructured communication does almost so. However, regular grid communications and global reduction operations continue to speedup as the subgrid length increases. In view of this, for our algorithm to make good use of the resources on the CM-5, a subgrid length of 512 or higher is required.

# Nodes	Grid Size	Node Total	Node CPU	Communication		
				NEWS	Send/Get	Reduction
32	256x256	275.3	169.8	63.7	31.7	10.1
32	512x512	963.7	709.7	118.0	112.4	23.6
64	512x512	504.7	352.0	77.9	58.8	16.0
128	512x512	289.1	175.2	69.2	32.0	12.8

Table 5: Scaling data for the **VF** algorithm using the **UC4** communication strategy.

In summary, we have designed an efficient semi-Lagrangian algorithm for shallow fluid flows representative of geophysical motions and shown that its implementation on the Connection Machine is almost fully scalable. Familiarity with the available software utilities on the CM-5 allowed us to significantly improve the computational efficiency of the model.

Acknowledgements

We would like to thank the Advanced Computing Laboratory at the Los Alamos National Laboratory for time and system support on the CM-5. This work was supported in part by the U.S. Department of Energy's CHAMMP program.

References

1. Jiang, S., Jin F-F, and Ghil, M., 1995: Multiple equilibria, periodic, and aperiodic solutions in a wind-driven, double-gyre, shallow-water model. *J. Phys. Ocean.*, **25**, 764-786.

2. Smolarkiewicz, P. K., and Pudykiewicz, J. A., 1992: A class of semi-Lagrangian approximations for fluids. *J. Atmos. Sci.*, **49**, 2082–2096.

3. Green, A. E., and Naghdi, P. M., 1976: A derivation of equations for wave propagation in water of variable depth. *J. Fluid Mech.*, **78**, 237–246.

4. Smolarkiewicz, P. K., and Margolin, L. G., 1994: Variational solver for elliptic problems in atmospheric flows. *Appl. Math. and Comp. Sci.*, **4**, 101–125.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A Reduced Grid Model For Shallow Flows on the Sphere

J.M. Reisner[a] and L.G. Margolin[a]

[a]Los Alamos National Laboratory, Los Alamos, New Mexico

P.K. Smolarkiewicz[b]

[b] National Center for Atmospheric Research, Boulder, Colorado

We describe a numerical model for simulating shallow water flows on a rotating sphere. The model is augmented by a reduced grid capability that increases the allowable time step based on stability requirements, and leads to significant improvements in computational efficiency. The model is based on finite difference techniques, and in particular on the nonoscillatory forward-in-time advection scheme MPDATA. We have implemented the model on the massively parallel CM-5, and have used it to simulate shallow water flows representative of global atmospheric motions. Here we present simulations of two flows, the Rossby-Haurwitz wave of period four, a nearly steady pattern with a complex balance of large and small scale motions, and also a zonal flow perturbed by an obstacle. We compare the accuracy and efficiency of using the reduced grid option with that of the original model. We also present simulations at several levels of resolution to show how the efficiency of the model scales with problem size.

1. Introduction

We describe a numerical model for simulating shallow water flows on a rotating sphere. The model is based on Eulerian spatial differencing and nonoscillatory forward-in-time (NFT) temporal differencing. Finite difference methods have advantages over spectral methods when implemented on massively parallel computers with distributed memory because the computations are localized in space. However finite difference methods have more restrictive time step constraints and so computational efficiency becomes an important issue. Our model is explicit in time, and its computational time step is limited by the largest Courant number on the mesh. Since the fastest wave speed is that of gravity waves, and is uniform over the mesh, the largest Courant number is associated with the smallest cell dimensions. In the typical latitude-longitude mesh, these smallest dimensions are found in the cells nearest the poles.

There are several strategies available to increase the time step and thus improve the computational efficiency of a finite difference model. For example, we have developed a semi-implicit version of our model (Nadiga et al. 1996 and Smolarkiewicz and Margolin 1994) in which the gravity waves are treated implicitly, but the advective velocities (which are much slower) are treated explicitly. In typical atmospheric applications, semi-implicit methods may allow a four-fold increase in time step. However the semi-implicit formulation leads to an elliptic problem, whose solution involves inverting a matrix on the mesh. Furthermore, the matrix operator is not symmetric due to the presence of Coriolis forces. This means that standard conjugate gradient methods may not converge and less optimal methods must be explored. Another alternative to allow larger time steps is filtering the velocities at the high latitudes. Filtering, however, requires global communication, making application on a massively parallel computer with distributed memory very inefficient.

Yet another alternative is the reduced grid. Here the logical structure of the regular latitude-longitude mesh is modified by removing some of the cells near the poles, effectively making the remaining cells larger. For example, if every other cell is removed from the regions within 30° of the poles, then the time step can be increased by a factor of two. The reduced grid also requires some nonlocal communication, but its impact on efficiency is much smaller than that of filtering. In addition, in our experiments we have found that the reduced grid reduces the accuracy less than either filtering or using implicit techniques.

In the following sections, we will briefly describe the model and the the reduced grid. We will then show results for two problems from the suite of test problems described by Williamson et al. 1992; these are the Rossby-Haurwitz wave and the steady-state zonal flow on the sphere. The use of larger time-steps reduces the numerical diffusion associated with truncation error. In the first case this is the dominant source of error, so that the reduced grid model yields more accurate results than the nonreduced grid. In the second case the interpolation errors dominate the diffusion errors and the reduced grid model is less accurate than the nonreduced model. We will provide error estimates for the two cases, as well as timing statistics for both the reduced and nonreduced grid in section 4. We summarize our results in section 5.

2. Shallow Water Model

The equations expressing conservation of mass and momentum for a shallow fluid on a rotating sphere are as follows:

$$\frac{\partial G\Phi}{\partial t} + \nabla \bullet (\mathbf{v}\Phi) = 0, \tag{1a}$$

$$\frac{\partial GQ_x}{\partial t} + \nabla \bullet (\mathbf{v}Q_x) = GR_x, \tag{1b}$$

$$\frac{\partial GQ_y}{\partial t} + \nabla \bullet (\mathbf{v}Q_y) = GR_y, \tag{1c}$$

where $G = h_x h_y$, and h_x and h_y represent the metric coefficients of the general orthogonal coordinate system, $\Phi = H - H_o$ is the thickness of the fluid with H and H_o denoting the height of the free surface and the height of the bottom, $\mathbf{v}$ is the horizontal velocity vector, and $\mathbf{Q} = (\Phi u h_x, \Phi v h_y)$ is the momentum vector. The right-hand-side forcings are

$$R_x = -\frac{g}{h_x}\Phi\frac{\partial(\Phi + H_o)}{\partial x} + fQ_y + \frac{1}{G\Phi}\left(Q_y\frac{\partial h_y}{\partial x} - Q_x\frac{\partial h_x}{\partial y}\right)Q_y \tag{2a}$$

$$R_x y - \frac{g}{h_y}\Phi\frac{\partial(\Phi + H_o)}{\partial y} - fQ_x + \frac{1}{G\Phi}\left(Q_y\frac{\partial h_y}{\partial x} - Q_x\frac{\partial h_x}{\partial y}\right)Q_x, \tag{2b}$$

where g is the acceleration of gravity and f is the Coriolis parameter.

The integration in time of the discretized approximations to (1) is described in Smolarkiewicz and Margolin (1993). The basic element of our nonoscillatory forward-in-time (NFT) algorithm is the sign-preserving advection scheme MPDATA (Smolarkiewicz 1984). The use of two-time-level integration schemes is a departure for Eulerian global atmospheric models where three-time-level or leapfrog schemes are traditionally used. However two-time-level schemes are widely preferred in most other fields of computational fluid dynamics. Some of the advantages of NFT schemes include a larger computational time step, reduced memory usage, and less numerical dispersion. In addition, the nonoscillatory property is crucial for preserving the sign of the layer thickness and of the thermodynamic scalars, and further controls the nonlinear stability of the computations. The model is implemented on a rotating sphere, and allows for arbitrary bottom topography as well as a free surface on top of the layer.

We have ported the model to the CM-5. It runs in data parallel mode, with the horizontal dimensions being spread across processors. In a typical problem, the speed

(measured in Megaflops) depends on problem size. For 32 nodes, a problem with a 64x128 mesh yields performance equivalent to 0.5 CRAY YMP processors, whereas a problem with 256x512 nodes runs at a speed equivalent to 1.5 CRAY YMP processors.

The reduced grid that we have implemented is adapted from the work of Rasch (1994). We use a sequence of nonoverlapping domains, where the number of grid points along circles of latitude decreases as one approaches the poles. The number of points from one domain to the next decreases by multiples of two, both for accuracy of interpolation as well as efficiency on the CM-5 architecture. One difference from Rasch (1994) is that the minimum number of grid points at a given latitude that is allowed for the top domains was fixed at 32 and not 4. This choice results both in increased accuracy and efficiency. Initially the latitude at which the reduction occurred was chosen as suggested by Rasch (1994); however sensitivity tests have revealed that the most accurate results occur when the reductions occur only in the vicinity of the pole—with only a slight increase in CPU time (with 2.8° resolution about 1 s for 7 days)—of a simulation being noted with this strategy. For example, in a simulation with resolution of 2.8° resolution at the equator three grids of 128×58×1, 64×4×2, and 32×4×2 are used to cover the globe (see Fig. 1 for a visual representation of the reduced grid). The ghost points at the top and bottom of each domain are simply interpolated values from other domains to be used in the calculations of north-south derivatives within a given domain. We use our NFT advection scheme for this interpolation, thus maintaining the positivity of the height and thermodynamic fields (Smolarkiewicz and Grell, 1992) and preserving the overall second-order accuracy of the model. This interpolation does require some nonlocal communication. At present, no temporal interpolation is used between the meshes of the reduced grid. The ratio between the time step required for the regular grid versus the reduced grid ranges from 4 with 2.8° resolution to 50 with 0.35° resolution. Note that as the resolution is increased in the nonreduced grid, the velocity field becomes more finely resolved, and locally may exceed the maximum values found in the less resolved solution. To ensure stability in such cases, the time step must be reduced by more than a factor of 4 as we increase the nonreduced grid's resolution from 0.7° to 0.35°—see Rasch (1994). We will demonstrate in section 4 that the nonlocal communication associated with interpolation does not allow for a corresponding ratio of CPU time between the reduced versus nonreduced grid on the CM5.

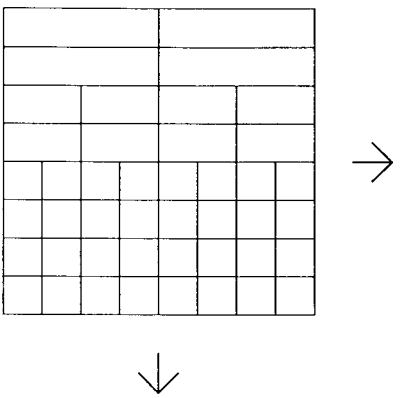


Fig. 1 Upper left portion of reduced grid.

3. Shallow Water Test Problems and Model Setup

Shallow water is a useful testbed for evaluating and comparing numerical methods that can be extended to fully three-dimensional global circulation models (GCMs). Also, shallow water layers can be stacked, and with the addition of a hydrostatic equation and slight recoding of (1)-(2), can be extended to model fully three-dimensional flows (cf. Bleck 1984). Such models are termed isentropic in meteorology. As part of DOE's CHAMMP program, Williamson et al. (1992) have published a suite of seven test problems for shallow water. One of these, the Rossby Haurwitz (RH) wave is particularly challenging, since it represents an almost steady state flow that is a delicate balance of large and small spatial scales. (When the free surface is replaced by a rigid lid, the solution is an exact steady state.) The RH wave is also interesting because it represents the advection of a wave which may be significantly damped if low-order forward-in-time methods are used (Smolarkiewicz and Margolin, 1993). Thus, the use of a larger time step reduces the numerical dissipation so that the reduced grid may perform as well or better than the regular grid.

Another test of the robustness of the reduced grid is a zonal flow over the pole. We have modified Williamson's standard test by the inclusion of a 6 km tall mountain in an approximately 8 km depth fluid. The mountain is located on the north pole. Although the mountain introduces an unsteady component into the problem, the flow is still relatively balanced and little change of position of the height contours is noted during a simulation. Unlike the RH wave in which dissipation errors dominate interpolation errors, interpolation errors dominate this flow situation. Thus the total error in the reduced grid formulation is found to be larger than in the nonreduced grid. Another source of error in the reduced grid formulation is that the topography on the coarser meshes found near the pole is not as well-resolved as in the nonreduced grid, potentially leading to additional differences between the two solutions. Because analytic solutions are not known for either the RH wave or the perturbed zonal-flow case, we have run high resolution simulations as a standard for comparing the results from the nonreduced and reduced grid. Except for timing statistics, all results are from simulations with 2.8° resolution at the equator. The simulations were run for a period of 7 days with a time step of 160/40 s being used for the reduced/nonreduced grid.

4. Results

4.1. Rossby-Haurwitz wave

Since visual differences between the reduced grid and nonreduced grid are not discernible, and the solutions have been published elsewhere (Smolarkiewicz and Margolin 1993, Fig. 2), we show only the L_2 and L_∞ error measures (see Fig. 2a) with respect to the height of the shallow fluid as a function of time for the reduced and nonreduced grids. These are the measures of error recommended by Williamson et al. (1992). Even in the error comparisons, very little difference is apparent—the errors associated with the reduced grid are only slightly smaller than those of the nonreduced grid. A further test was conducted in which the reduced grid's time step was halved (160s to 80s) to determine the sensitivity of the solution to numerical dissipation. As expected, the errors fall in between those of the nonreduced grid and of the reduced grid that used a twice-larger time step.

Our original motivation for introducing the reduced grid was to improve computational efficiency. Table 1 demonstrates that with 2.8° resolution the total CPU time (total time is for one hour of simulated time) for the nonreduced grid (denoted by 2.8

in Table 1) and for the reduced grid (denoted by 2.8r in Table 1) are nearly equal. The ratio increases to about 35 with 0.35° resolution. Hence, at least on the CM5, the bigger gains in efficiency occur for the higher resolution grids. The main cause for the increasing ratio with grid resolution is directly related to the ratio of time steps required for each approach, which increases with decreasing resolution (see discussion at the end of section 2). Breaking down the total CPU time into four components, node CPU time, NEWS or parallel communication (e.g., cshifts), Send/Get or nonparallel communication (e.g., interpolation), and other (e.g., communication between nodes and and program manager, global sums, ect...) we observe that for the nonreduced grid the primary component is NEWS; whereas for the reduced grid the primary component is Send/Get for the smaller domains and on node calculations for the larger domains. In addition, the reduced grid contains fewer grid points than the nonreduced grid, so that the amount of memory used in the larger grids of the reduced mesh is somewhat less than that of the nonreduced mesh (about 100 mb for a grid with 0.35° resolution).

Table 1

Resolution°	Node CPU	NEWS	Send/Get	Other	Total
2.8	0.864	2.955	0.000	0.009	3.828
2.8r	0.984	1.807	2.552	0.216	5.559
1.4	8.684	26.504	0.000	0.052	35.240
1.4r	4.152	5.076	7.800	1.932	18.960
0.7	119.504	325.216	0.000	0.336	445.056
0.7r	18.224	15.248	20.880	7.056	61.408
0.35	2860.100	6914.200	0.000	4.300	9778.600
0.35r	116.944	54.764	51.326	50.060	273.094

4.2. Zonal Flow

Unlike the RH wave, visual differences are apparent between the solutions produced by the reduced and nonreduced meshes for the perturbed zonal flow. Figs. 3a, 3b, and 3c, show the numerical solutions for the reduced mesh, the nonreduced mesh, and a high resolution simulation. Again we quantify the differences of these three solutions in terms of the L_2 and L_∞ error measures (see Fig. 2b) of the thickness of the fluid. In comparison with the errors of the RH wave, the absolute error levels for this case with respect to the highly resolved simulation are smaller—the L_2 error is about an order of magnitude smaller. Hence, as was noted in Smolarkiewicz and Margolin (1993) the smaller error measures for this case suggest a less dissipative flow regime (due to numerical discretization). The fact that the error measures for the reduced grid are greater than the nonreduced grid also suggest that the main cause of these differences is due to errors associated with the interpolation in the reduced grid.

As noted in section 2 a shallow-water model can be easily extended to include the effects of baroclinicity in the atmosphere. To investigate whether adding additional layers, and hence requiring additional interpolation will degrade the quality of the solution we ran the zonal flow problem with 8 layers for both the reduced grid and the nonreduced grid. Our analysis of the results of these runs indicate that adding additional layers to the reduced grid model does not further degrade the solution with respect to the nonreduced grid. In addition, the timing statistics suggest that as the number of levels in the vertical increases, the efficiency of the reduced grid also increases, so that with 2.8° resolution and 64 layers the reduced grid is about 1.25 times faster than the nonreduced grid.

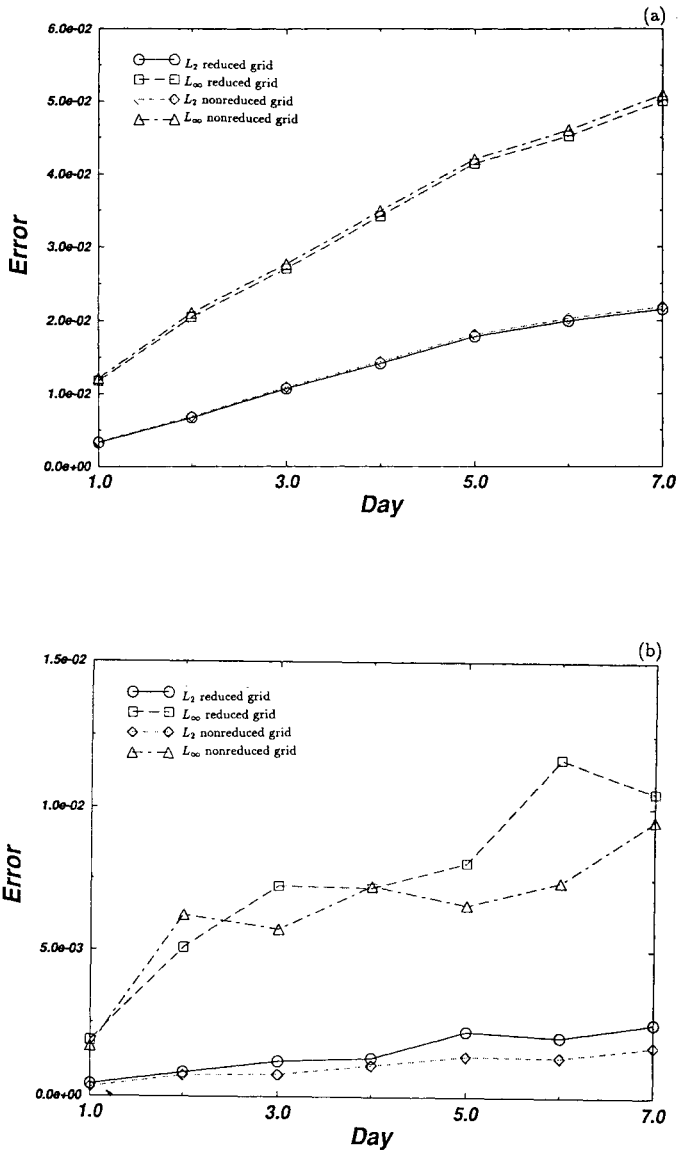


Fig. 2 Time evolution of the error measures for the model configuration with 2.8° resolution for the (a) Rossby-Haurwitz wave and for (b) Zonal Flow.

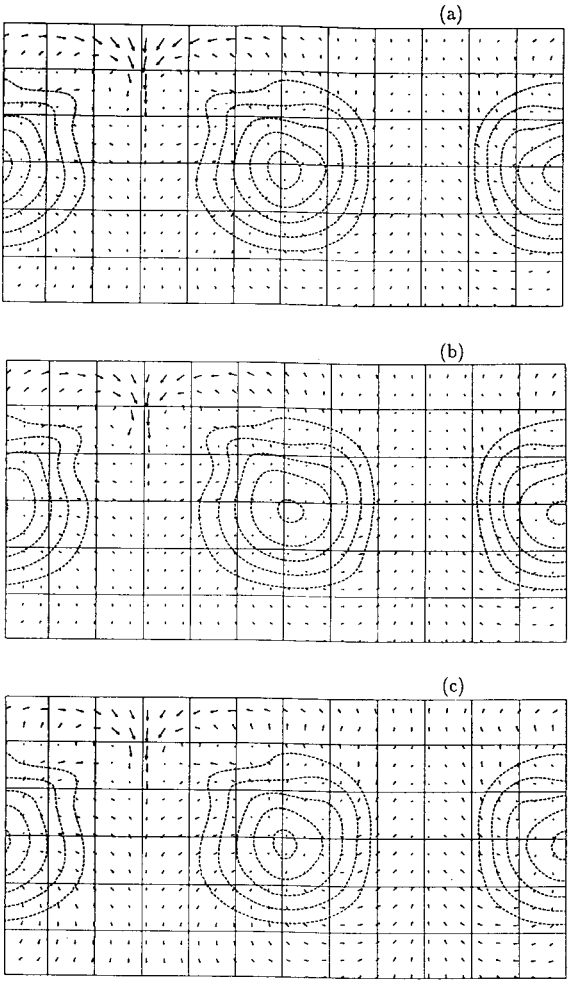


Fig. 3 Geopotential perturbation and the flow vectors for the global nonlinear zonal geostrophic flow test problem with a tall mountain at the pole: (a) the reduced mesh; (b) the nonreduced mesh; and (c) the resolved solution.

5. Conclusions

We have described a numerical model for shallow flows on the sphere. We have developed a reduced grid model to circumvent the time step constraints, and consequent lack of efficiency associated with the convergence of meridians at the poles. At low resolution we have shown that the use of the reduced grid is as accurate as the nonreduced grid with about the same computational efficiency. The principal computational advantage of the reduced grid is realized at higher resolutions, with the reduced grid being up to 35 times faster than the nonreduced grid at a resolution of 0.35°. Due to faster communication on machines like the Cray T90 or T3D we believe that the reduced grid will lead to greater efficiency on these machines. Thus we conclude that the reduced grid framework is a useful modification for general circulation models of the atmosphere and ocean based on finite difference approximations.

6. Acknowledgements

The authors appreciate continuing discussions with Phil Rasch, and the help of the system programmers of the Los Alamos Advanced Computing Laboratory. This work was supported by in part by the CHAMMP program of the U.S. Department of Energy.

REFERENCES

Bleck, R., 1984: An isentropic model suitable for lee cyclogenesis simulation. *Riv. Meteor. Aeronaut.*, **43**, 189-194.

Nagida, B.T., L.G. Margolin, and P.K. Smolarkiewicz, 1995: Semi-Lagrangian Shallow Water Modeling on the CM5. submitted to special issue of Elsevier Science Publishers for Proceedings of Parallel CFD'95.

Rasch, P.J., 1994: Conservative shape-preserving two-dimensional transport on a spherical reduced grid. *Mon. Wea. Rev.*, **122**, 1337-1350.

Smolarkiewicz, P.K., 1984: A fully multidimensional positive definite advection transport algorithm with small implicit diffusion. *J. Comp. Phys.*, **54**, 323-362.

Smolarkiewicz, P.K. and G.A. Grell, 1992: A class of monotone interpolation schemes. *J. Comp. Phys.*, **101**, 431-440.

Smolarkiewicz, P.K. and L.G. Margolin, 1993: On forward-in-time differencing for fluids: extension to a curvilinear framework. *Mon. Wea. Rev.*, **121**, 1847-1859.

Smolarkiewicz, P. K., and L. G. Margolin, 1994: Forward-in-time differencing for global dynamics. In *Preprint Vol. Fifth Symposium on Glabal Change Studies*, 23-28 January, Nashville, Tennesse, American Meteorological Society, 99.

Williamson, D.L., J.B. Drake, J.J. Hack, R. Jakob, and P.N. Swarztrauber, 1992: A standard test set for numerical approximations to the shallow water equations on the sphere. *J. Comp. Phys.*, **102**, 211-224.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Application of a Parallel Navier-Stokes Model to Ocean Circulation

Chris Hill and John Marshall

Department of Earth, Atmospheric and Planetary Sciences,
Massachusetts Institute of Technology

Abstract

A prognostic ocean circulation model based on the incompressible Navier-Stokes equations is described. The model is designed for the study of both non-hydrostatic and hydrostatic scales; it employs the "pressure method" and finite volume techniques. To find the pressure in the non-hydrostatic limit, a three-dimensional Poisson equation must be inverted. We employ a single-pass, two-grid method where the thin three-dimensional domain of the ocean is first projected on to a two-dimensional grid using vertical integration as a smoother. The two-grid method exploits the anisotropic aspect ratio of the computational domain and dramatically reduces the computational effort required to invert the three-dimensional elliptic problem. The resulting model is competitive with hydrostatic models in that limit and scales efficiently on parallel computers.

1 Introduction

The ocean is a stratified fluid on a rotating body driven from its upper surface by patterns of momentum and buoyancy flux. The detailed dynamics are very accurately described by the Navier-Stokes equations. These equations admit, and the ocean contains, a wide variety of phenomenon on a plethora of space and time scales. Modeling of the ocean is thus a formidable challenge; it is a turbulent fluid containing energetically active scales ranging from the global down to order 1 - 10 km horizontally and some 10's of meters vertically - see Fig.1. Important scale interactions occur over the entire spectrum.

Numerical models of the ocean circulation, and the ocean models used in climate research, employ approximate forms of the Navier-Stokes equations. Most are based on the "hydrostatic primitive equations" (**HPE**) in which the vertical momentum equation is reduced to a statement of hydrostatic balance and the "traditional approximation" is made in which the Coriolis force is treated approximately and the shallow atmosphere approximation is made. On the large-scale the terms omitted in the **HPE** are generally thought to be small, but on small scales the scaling assumptions implicit in them become increasingly problematic.

In this paper we report on an ocean model firmly rooted in the incompressible Navier-Stokes (**INS**) equations. The novelty of our approach is that it is sensitive to the transition from non-hydrostatic to hydrostatic behavior and performs competitively with hydrostatic models in that limit – more details can be found in Marshall et.al. [10], [9] and Arvind et.al [1].

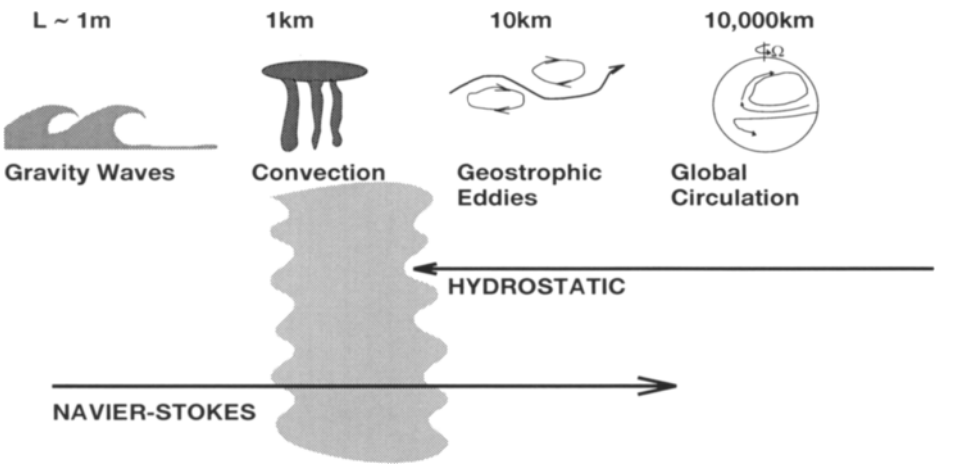


Figure 1: *A schematic diagram showing the range of scales in the ocean. Global/basin scale phenomenon are fundamentally hydrostatic, convective processes on the kilometer scale fundamentally non-hydrostatic. Somewhere between 10km and 1km the hydrostatic approximation breaks down – the gray area in the figure.* **HPE** *schemes are designed to study large-scale processes but are commonly used at resolutions encroaching on the "gray area" – the left-pointing arrow.* **INS** *is valid across the whole range of scales. We show in this paper that the computational overhead of* **INS** *is slight in the hydrostatic limit and so* **INS** *can also be used with economy for study of the large-scale – the right-pointing arrow.*

Our algorithm solves **INS** on the sphere building on ideas developed in the computational fluids community. Finite volume methods are used which makes possible a novel treatment of the boundary in which volumes abutting the bottom or coast may take on irregular shapes and so be "sculptured" to fit the boundary. The numerical challenge is to ensure that the evolving velocity field remains non-divergent. Most procedures, including the one employed here, are variants on a theme set out by Harlow and Welch [6], in which a pressure correction to the velocity field is used to guarantee non-divergence. The correction step is equivalent to, and is solved as, a three-dimensional Poisson problem for the pressure field with Neumann boundary conditions. A "brute force", unthinking approach to the Poisson inversion requires prohibitive amounts of computer time, and would render an **INS** model which, even in its hydrostatic limit, is uncompetitive with **HPE**. The inversion demands "global" connectivity between all points in the computational domain, presenting a challenge in mapping the model to a parallel computer because the connectivity requires communication right across the grid to the boundary. A major objective of our study, therefore, has been to design a Poisson solver that was efficient and could map well to parallel architectures, thereby making **INS** a powerful tool applicable to all scales of interest in oceanography.

In our approach the pressure field is separated in to surface, hydrostatic and non-hydrostatic components and the elliptic problem is solved in two stages. First, as in hydrostatic models, a two-dimensional elliptic equation is inverted for the surface pressure. Then, making use of the surface pressure field thus obtained, the non-hydrostatic pressure is found through inversion of a three-dimensional problem. Preconditioned conjugate gradient iteration is used to invert the symmetric elliptic operators in both two and three dimensions. Our method exploits the fact that as the horizontal scale of the motion becomes very much larger than the vertical scale, the motion becomes more and more hydrostatic and the three-dimensional elliptic operator becomes increasingly anisotropic and dominated by the vertical axis. Accordingly a preconditioner for the three-dimensional problem is used which, in the hydrostatic limit, is an exact integral of the elliptic operator, and so leads to a single algorithm that seamlessly moves from non-hydrostatic to hydrostatic limits. Thus, in the hydrostatic limit, the model is "fast", competitive with the fastest ocean climate models in use today based on the hydrostatic primitive equations. But as the resolution is increased the model dynamics asymptote smoothly to the Navier-Stokes equations and so can be used to address small-scale processes.

2 The numerical strategy

2.1 Time Stepping

We write the Boussinesq incompressible Navier Stokes equations describing our ocean model in semi-discrete form to second order in the time Δt, in which only time is discretized:

$$\frac{\underline{v}_h^{n+1} - \underline{v}_h^n}{\Delta t} = \underline{G}_{v_h}^{n+\frac{1}{2}} - \nabla_h \left\{ p_s + p_{hy} + q p_{nh} \right\}^{n+\frac{1}{2}} \tag{1}$$

$$\frac{w^{n+1} - w_n}{\Delta t} = \widehat{G}_w^{n+\frac{1}{2}} - \frac{\partial p_{nh}^{n+\frac{1}{2}}}{\partial z} \tag{2}$$

$$\frac{\partial w^{n+1}}{\partial z} + \nabla_h . \underline{v}_h^{n+1} = 0 \tag{3}$$

Equations (1)–(3) describe the time evolution of the flow; $\underline{v} = (\underline{v}_h, w)$ is the velocity in the horizontal and the vertical, and the $\underline{G}$ terms incorporate inertial, Coriolis, metric, gravitational, and forcing/dissipation terms.

In oceanographic applications equations (1)–(3) are solved in the irregular geometry of ocean basins in a rotating reference frame using a spherical polar coordinate system. The velocity normal to all solid boundaries is zero, and, in order to filter rapidly propagating surface gravity waves, we place a rigid lid is placed at the upper surface. (For brevity we have not written down equations for temperature and salinity which must also be stepped forward to find, by making use of an equation of state, the density ρ).

In (1) the pressure has been separated into surface, hydrostatic and non-hydrostatic components thus:

$$p(\lambda, \phi, z) = p_s(\lambda, \phi) + p_{hy}(\lambda, \phi, z) + q p_{nh}(\lambda, \phi, z) \tag{4}$$

where (λ, ϕ, z) are the latitude, longitude and depth respectively. The first term, p_s, is the surface pressure (the pressure exerted by the fluid under the rigid lid); it is only a function of horizontal position. The second term, p_{hy}, is the hydrostatic pressure defined in terms of the weight of water in a vertical column above the depth z,

$$\frac{\partial p_{hy}}{\partial z} + g' = 0 \tag{5}$$

where g' is the "reduced gravity" $g\frac{\delta\rho}{\rho_{ref}}$, and $\delta\rho$ is the departure of the density from a reference profile, and ρ_{ref} is a constant reference density. The third term is the non-hydrostatic pressure, p_{nh}. Note that in (1) and (4) we employ a "tracer " parameter q which takes the value zero in **HPE** and the value unity in **INS**. In **INS** w is found using (2), from which large and balancing terms involving the hydrostatic pressure and gravity (5) have been canceled (and the remaining terms represented by $\widehat{G}_w$) to ensure that it is well conditioned for prognostic integration. In **HPE**, (2) is not used; rather w is diagnosed from the continuity relation (3). In the hydrostatic limit only p_{hy} and p_s are required in (4). Equation (5) is used to find p_{hy} and p_s is found by solving a two-dimensional elliptic problem for the surface pressure that ensures non-divergent depth-integrated flow:

$$\nabla_h . H \nabla_h p_s = \nabla_h . \overline{\underline{G}_{v_h}^{n+\frac{1}{2}}}^H - \nabla_h . \overline{\nabla_h p_{hy}}^H \tag{6}$$

($\overline{}^H$ indicates the vertical integral over the local depth, H, of the ocean and the subscript $_h$ denotes horizontal).

In **INS** q is set equal to unity and p_{nh} must be determined. It is found by inverting a three-dimensional elliptic equation ensuring that the local divergence vanishes:

$$\nabla^2 p_{nh} = \nabla . \underline{G}^{n+\frac{1}{2}} - \nabla_h^2 (p_s + p_{hy}). \tag{7}$$

where ∇^2 is a full three-dimensional Laplacian operator.

3 Finding the pressure

In addition to the two-dimensional inversion required in **HPE** (for example in Dukowicz [5] or Bryan [4]) **INS** requires a three-dimensional elliptic problem to be solved; this is the overhead of **INS** relative to **HPE**.

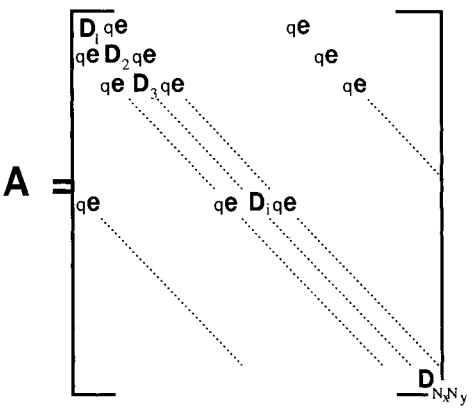


Figure 2: *The structure of the three-dimensional Laplace matrix* **A**.

3.1 The three-dimensional elliptic problem.

Equation (7) can be written in discrete form as a set of simultaneous equations using matrix-vector notation:

$$\mathbf{A}p = f \tag{8}$$

A is a symmetric, positive-definite matrix representing the discrete form of ∇^2 in three-dimensions. It is a sparse seven-diagonal operator with coefficients that depend on the geometry of the ocean basin.

We use a preconditioned conjugate gradient scheme (PCG) to solve (8). Figure 2 shows the form of **A**. Each block, **D**, in the figure is a tri-diagonal matrix representing $\frac{\partial^2}{\partial z^2}$ while **e** represents the off-diagonal terms from the discretized ∇_h^2 operator. If the vertical scale is much smaller than the horizontal scale then **A** is dominated by the **D** blocks along its central diagonals; the elements **e** are $\frac{\Delta z^2}{\Delta x^2}$ smaller than elements of **D**. Moreover, the tracer parameter q from (1) and (4) always appears as a multiplier of **e**; thus when q is set to zero (corresponding to the hydrostatic limit), **A** is comprised *only* of the blocks **D** and so can readily be inverted. Accordingly we use a preconditioner during the inversion of **A** which is the exact inverse of the blocks, **D**, and which is thus an exact inverse of **A** in the hydrostatic limit. Rather than store the block inverses, the preconditioner uses LU decomposition.

4 Numerical implementation

The algorithm outlined in the previous sections was developed and implemented on a 128-node CM5, a massively-parallel distributed-memory machine in the Laboratory for Computer Science (LCS) at MIT. The code was written in CMFortran, a data-parallel FORTRAN, closely related to High Performance Fortran (HPF). The algorithm was also coded in the implicitly parallel language Id, permitting a multithreaded implementation on MIT's data flow machine MONSOON. The programming issues are developed

more fully in Arvind et.al.[1], where implicitly parallel multithreaded and data parallel implementations are compared.

In deciding how to distribute the model domain over the available processors on a distributed memory machine, we had to bear in mind that the most costly task in our algorithm is finding the pressure field. Our PCG procedure entails nearest-neighbor communication between adjacent grid points in the horizontal and communication over all points in the vertical. Accordingly we decompose the domain in to vertical columns that reach from the top to the bottom of the ocean. In this way the computational domain and workload are distributed across processors laterally in equally sized rectangles.

4.1 Model Performance

In the hydrostatic limit our scheme performs comparably with conventional second-order finite difference **HPE** codes. In typical applications the model has many vertical levels and most of the computer time is spent evaluating terms in the explicit time-stepping formulae; the work required to invert the two-dimensional elliptic problem does not dominate. In **INS**, however, the computational load can be significantly increased by the three-dimensional inversion required. One iteration of the three-dimensional elliptic solver is equivalent to 6% of the total arithmetic operation count of the explicit time stepping routines. Therefore, if the three-dimensional inversion is not to dominate the prognostic integration, the PCG procedure must converge in less than fifteen iterations.

Solving for the pressure terms in (4) separately, rather than all at once, dramatically improves the performance of **INS**. We illustrate this here in the case of a high resolution simulation of convective overturning (reported in detail in Marshall et.al. [10]). The aspect ratio of the grid cells is of order 0.2, thus allowing significant non-hydrostatic effects. However, $\mathbf{A}$ is still dominated by the blocks along its diagonal. To demonstrate the benefit of splitting the pressure terms we took the right hand side from a single time step of the model and solved (7) for p_{nh} using increasingly accurate p_s solutions from (6). The results are summarized in figure 3.

Fig. 3 shows, as a surface, the number of iterations, I_{3d}, required by our PCG solver for (8) to reach a given accuracy of solution, r_{3d}. The surface is a measure of the convergence rate of the three-dimensional solver; its shape shows how the convergence rate depends on the accuracy of the solution for p_s from (6). To generate fig. 3 we varied the accuracy, r_{2d}, to which we solved (6), and recorded the resulting iteration count to reach a given r_{3d}.

In the physical scenario studied values of $r_{3d} < 10^{-7}$ are required to keep the flow field sufficiently non-divergent and maintain numerical stability. From fig. 3 we see that if p_s is found inaccurately or not at all ($\log r_{2d} = 0$; we normalize the residual so that $r_{2d} = 1$ at the first iteration) then solving (8) to $r_{3d} \approx 10^{-7}$ will require some 350 three-dimensional inverter iterations, **at every timestep**. In contrast, if p_s is found accurately (i.e. $r_{2d} < 10^{-6}$, in the flat "valley" of the figure) then the number of three-dimensional inverter iterations drops to < 10 and the overhead of **INS** relative to **HPE** is not prohibitive. Note, the change in the slope of the I_{3d} surface indicates that the drop in iteration count is due to accelerated convergence and is not a consequence of an

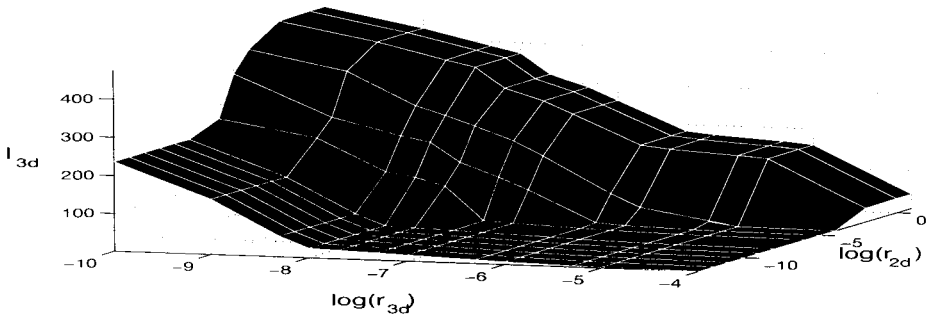


Figure 3: *Variation in convergence rate of our three-dimensional PCG solver as a function of the accuracy of the p_s term on the right hand side of equation (7). The convergence rate improves drastically when p_s is found accurately.*

improved initial guess.

5 Conclusions

On a 512 node CM5 our code achieves a sustained floating-point throughput of 10 GFlop/sec. With 100+ GFlop/sec computational capability available in the near future we will be able to model the ocean at increasingly fine resolutions.The model discussed here scales efficiently, and will be able to exploit such parallel machines as they become available. Because the dynamics is rooted in **INS** and the algorithm is designed to handle small scales, it is well suited to simulations of the ocean at very high spatial resolution. A major objective of our study has been to make models based on **INS** efficient and therefore usable. The "split solver" methodology developed renders **INS** a feasible approach and could be applied to many three-dimensional CFD problems which are spatially anisotropic with a "preferred" dimension.

The possible applications of such a model are myriad. In the hydrostatic limit it can be used in a conventional way to study the general circulation of the ocean in complex geometries. But because the algorithm is rooted in **INS**, it can also address (for example) (i) small-scale processes in the ocean such as convection in the mixed layer (ii) the scale at which the hydrostatic approximation breaks down (iii) questions concerning the posedness of the hydrostatic approximation raised by, for example, Browning et.al.[2] and Mahadevan et.al.[7],[8]. Finally it is interesting to note that - as described in Brugge et.al.[3] - the incompressible Navier Stokes equations developed here are the basis of the pressure-coordinate, quasi-hydrostatic atmospheric convection models developed by Miller[12],[11]. Thus the ocean model described here is isomorphic to atmospheric forms suggesting that it could be developed, and coupled to, a sibling atmospheric model based on the same numerical formulation.

6 Acknowledgments

This research was supported by ARPA and TEPCO. The MIT CM5 is part of the SCOUT initiative led by Tom Green and Al Vezza. Advice and encouragement on computer science issues was given by Arvind of LCS. J. White of the Department of Electrical Engineering and Computer Science at MIT advised on the solution of elliptic problems. A. White of the UK Meteorological Office consulted on the formulation of the model.

References

[1] Arvind, K-C Cho, C. Hill, R. P. Johnson, J. C. Marshall, and A. Shaw. A comparison of implicitly parallel multithreaded and data parallel implementations of an ocean model based on the Navier-Stokes equations. Technical Report CSG Memo 364, MIT LCS, 1995.

[2] G.L. Browning, W.R. Holloand, H-O Kreiss, and S.J. Worley. An accurate hyperbolic system for approximately hydrostatic and incompressible flows. *Dyn. Atmos. Oceans*, pages 303–332, 1990.

[3] R. Brugge, H.L. Jones, and J.C. Marshall. Non-hydrostatic ocean modelling for studies of open-ocean deep convection. In *Proceedings of the Workshop on Deep convection and deep water formation in the ocean*, pages 325–340. Elsevier Science Publishers, Holland, 1991.

[4] K. Bryan and M.D. Cox. A nonlinear model of an ocean driven by wind and differental heating: Parts i and ii. *J. Atmos. Sci.*, 25:945–978, 1968.

[5] J. Dukowicz, R. Smith, and R.C. Malone. A reformulation and implementation of the Bryan-Cox-Semptner ocean model on the Connection Machine. *J. Ocean. Atmos. Tech.*, 10(2):195–208, 1993.

[6] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow. *Phys. Fluids*, 8:2182, 1965.

[7] A. Mahadevan, J. Oliger, and R. Street et.al. A non-hydrostatic mesoscale ocean basin model I: well-posedness and scaling. submitted *J. Phys. Oceanogr.*, 1995.

[8] A. Mahadevan, J. Oliger, and R. Street et.al. A non-hydrostatic mesoscale ocean basin model II: numerical implementation. submitted *J. Phys. Oceanogr.*, 1995.

[9] J. C. Marshall, A. Adcroft, C. Hill, and L. Perelman. A finite-volume, incompressible Navier-Stokes model for studies of the ocean on parallel computers. submitted *J. Geophys Res.*, 1995.

[10] J. C. Marshall, C. Hill, L. Perelman, and A. Adcroft. Hydrostatic, quasi-hydrostatic and non-hydrostatic ocean modeling. submitted *J. Geophys Res.*, 1995.

[11] M.J. Miller. On the use of pressure as vertical co-ordinate in modelling. *Quart. J. R. Meteorol. Soc.*, 100:155–162, 1974.

[12] M.J. Miller and R.P. Pearce. A three-dimensional primitive equation model of cumulonimbus convection. *Quart. J. R. Meteorol. Soc.*, 100:133–154, 1974.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A Portable and Efficient Parallel Code for Astrophysical Fluid Dynamics *

A. Malagoli, A. Dubey, F. Cattaneo [a] and D. Levine [b]

[a]EFI/LASR, University of Chicago
933 E.56th, Chicago, IL 60637, USA

[b]Argonne National Laboratory 9700 S. Cass. Ave, Argonne, IL 60439

We have developed an application code for scalably parallel architectures to solve the equations of compressible hydrodynamics for a gas in which the thermal conductivity changes as a function of temperature. The code has been developed to study the highly turbulent convective envelopes of stars like the sun, but simple modifications make it suitable for a much wider class of problems in astrophysical fluid dynamics. The algorithm consists of two components: (a) a finite difference *higher–order Godunov method* for compressible hydrodynamics, and (b) a Crank–Nicholson method based on nonlinear *multigrid* method to treat the nonlinear thermal diffusion operator. These are combined together using a time splitting formulation to solve the full set of equations. The code has been developed using the PETSc library (Gropp & Smith 1994) to insure portability across architectures without loss of efficiency. Performance and scalability measurements on the IBM SP–2 will be presented.

1. Introduction

We have developed an application code for scalably parallel, distributed memory architectures using new development tools to insure optimal performance without sacrificing portability. The code solves the equations of compressible hydrodynamics for a gas in which the thermal conductivity changes as a function of temperature.

The scientific motivation for such development is related to the study of the highly turbulent convective layers of late type stars like the Sun, in which turbulent mixing plays a fundamental role in the redistribution of many physical ingredients of the star, such as angular momentum, chemical contaminants, and magnetic fields. Although the simulation of solar conditions is well beyond conceivable computing resources for some time to come, given that the Reynolds number Re is in excess of 10^{12}, our past research (see e.g. Cattaneo *et al.* 1991) has shown that increase in the dynamical range of turbulence, by means of increasing the number of computationally available degrees of freedom, leads to develop better intuition for the real situation. This in turn requires that the code be capable of Gflops, and possibly Tflops performance.

*This work was supported by a NASA/ESS Grand Challenge HPCC grant.

The core of the computational algorithm consists of two independent components: (a) a finite difference *higher–order Godunov method* for compressible hydrodynamics, and (b) a Crank–Nicholson method based on nonlinear *multigrid* method to treat the nonlinear thermal diffusion operator. The two algorithms are combined together using a time–splitting technique. Both algorithms have a reasonably modular structure, which makes them easily reusable in a much wider range of CFD applications than the specific one we are interested in.

The algorithms have been designed and implemented in standard f77 and C, and using callable functions and subroutines from the PETSc (Portable Extensible Toolkit for Scientific computing) libraries of W.Gropp and B.Smith (1994). The PETSc library is a collection of higher level tools for the solution of partial differential equations on sequential and parallel machines. The PETSc library version we have used employs the portable lightweight Chameleon message–passing interface, now being replaced by the MPI standard, and parallelism is handled transparently to the user by setting appropriate data structures. For example, we have used extensively the BlockComm subset of PETSc, a collection of subroutines for solving finite difference equations on regular domain decomposed grids. BlockComm automatically generates the data exchange pattern to update the boundary grid zones as the solution is computed, saving the programmer the task of writing low–level message passing code.

Most of the architecture–specific implementations are handled within PETSc, and therefore our application code easily compiles and runs on various architectures, including so far the Intel Delta, Intel Paragon, IBM SP, and clusters of workstations. Because the Chameleon interface provides an extremely low overhead to the native (and most efficient) message–passing libraries, portability does not imply loss of performance.

The following sections are organizes as follows. In section 2 we present the physical model and the corresponding equations; in section 3 we describe the discretization method; in section 4 we describe the parallelization method; and finally in section 5 we summarize experiments of code performance and scalability on the IBM SP-2 at the Maui High Performance Computing Center.

2. Problem Description

The physical motivation for this work originates from the study of turbulent convection and mixing in highly stratified, compressible plasmas in stars (see e.g. Cattaneo *et al.* 1991, Bogdan *et al.* 1993). The outer layers of stars like the sun are in a state of vigorous convection. Turbulent motions within the deep convective envelope transport nearly all of the energy produced in the stable radiative interior outward to the tenuous regions of the stellar photosphere. Furthermore, penetrative motions from the convection zone extend some distance downward into the stably–stratified radiative interior, influencing the distribution of chemical composition and other physical properties throughout the star. The transition from the stably–stratified interior to the convectively unstable region is caused by a change in the radiative opacity with the density and temperature of the plasma, so that radiative diffusion eventually becomes inefficient and convection sets in to transport the energy.

Our concern is to study the properties of multi–dimensional, convective turbulence

in a strongly stratified medium, where motions can span multiple scale heights and the fluid viscosity is so small that a turbulent state develops. In order to make the problem tractable, several simplifications are introduced and an idealized model is considered. A plane–parallel layer of gas is confined vertically by two impenetrable, stress –free walls, and is stratified under the influence of a constant gravitational force g. The gas is heated from the lower wall, and the temperature at the upper wall is kept constant. This model represents a generalization of the classical Rayleigh–Benard convection cell in laboratory fluids. The gas is given initially a hydrostatic configuration in such a way that the upper half is convectively unstable and the lower part is stably stratified. This can be achieved by choosing and appropriate functional form for the coefficient of thermal conductivity. The initial condition is then perturbed, and eventually the upper part of the layer begins convecting. The equations governing the behavior of fluid are :

$$\partial\rho + \nabla \cdot \rho u = 0 \tag{1}$$

$$\partial_t \rho u + \nabla \cdot \rho u u = -\nabla P + g\rho z + Q_{visc} \tag{2}$$

$$\partial_t T + u \cdot \nabla T + (\gamma - 1) T \nabla \cdot u = \frac{1}{\rho C_v}[\nabla \cdot (\kappa(\rho, T)\nabla T)] + H_{visc} \tag{3}$$

where ρ, u and T are respectively the density, velocity and temperature of the gas; γ is the ratio of specific heats and C_v is the heat capacity at constant volume. Q_{visc} and H_{visc} represent the rates of viscous momentum dissipation and viscous heating. The coefficient of thermal conductivity is $\kappa(\rho, T)$, and in general it can be a function of ρ and T.

Results and animations from numerical simulations of this model can be found in our URL "http://astro.uchicago.edu/Computing".

3. The Numerical Scheme

We give here a description of the two–dimensional scheme, the extension to three–dimensions being straightforward. The fluid equations without the right–hand side of the energy equation (3) (the thermal conduction parabolic operator) form a system of hyperbolic equations, which we solve using the Piecewise Parabolic Method (PPM) of Colella and Woodward (1994) (see section 3.1)

In order to solve the full energy equation, we adopt a time–split formulation as follows:

- Solve the hydrodynamic equations (1), (2) and (3 without the rhs.) using a higher-order Godunov scheme to get $T^{n+\frac{1}{2}}_{hydro}$.
- Solve for the temperature diffusion with the intermediate temperature $T^{n+\frac{1}{2}}_{hydro}$ treated as a source term.

The time advance of the full energy equation 3 is therefore formulated with second–order accuracy in time in the form:

$$\frac{T^{n+1} - T^n}{\Delta t} = T^{n+\frac{1}{2}}_{hydro} + \frac{1}{2}\frac{\gamma C_k}{\rho}\nabla \cdot \left[\kappa(T^{n+1})\nabla T^{n+1} + \kappa(T^n)\nabla T^n\right] \tag{4}$$

Here, the thermal conduction term is treated implicitly with a Crank–Nicholson method to avoid overly restrictive stability constraints on the time step.

The spatial discretization for the entire scheme is done using a cell–centered formulation, where all the physical variables are defined at the cell centers, and the physical boundaries of the domain are along the cell edges (see figure 1). Godunov methods are intrinsically cell–centered schemes, since they are based on the conservation form of the hydrodynamic equations (1), (2) and (3) (see section 3.1). The nonlinear thermal conduction operator on the right-hand side of equation (4) is discretized as a standard, second–order accurate approximation that results in a finite difference operator with a 5 point stencil:

$$\begin{aligned} \nabla \cdot \kappa(T) \nabla T \approx & \frac{1}{2\Delta x^2}[(\kappa_{i+1\,j} + \kappa_{i\,j})(T_{i+1\,j} - T_{i\,j}) - (\kappa_{i\,j} + \kappa_{i-1\,j})(T_{i\,j} - T_{i-1\,j})] \\ & \frac{1}{2\Delta y^2}[(\kappa_{i\,j+1} + \kappa_{i\,j})(T_{i\,j+1} - T_{i\,j}) - (\kappa_{i\,j} + \kappa_{i\,j-1})(T_{i\,j} - T_{i\,j-1})] \end{aligned} \quad (5)$$

where $\kappa_i = \kappa(T_i)$.

The solution of the implicit scheme resulting from (4) and (5) requires the inversion of a nonlinear parabolic operator, for which we use a multigrid technique. Because the Crank–Nicholson scheme in (4) is unconditionally stable, the CFL condition required for the stability of the Godunov scheme is the only limiting factor in the choice of the integration step for the equations. Additional constraints may be imposed by accuracy considerations in cases in which the coefficient of thermal conduction κ is very large.

We now give a description of the higher-order Godunov method and of the nonlinear multigrid implementation.

3.1. Higher–order Godunov method

Higher-order Godunov methods have been used successfully for many years to solve problems in compressible fluid dynamics. The best representative of this class of schemes is certainly the Piecewise Parabolic Method (PPM) of Colella and Woodward (1984). These methods are typically second order accurate in space and time, and they remain robust even in the presence of discontinuities in the solution, such as shocks or contact discontinuities. This property makes PPM and related methods ideally suited to study problems involving compressible gases, as is the case in astrophysical plasmas. The method uses a cell–centered representation of the physical variables (density, temperature and velocity) on an Eulerian grid, and equations (1), (2), and (3) are cast in a system of conservation laws for the variables U = (mass, momentum, total energy). The method uses a symmetrized sequence of one–dimensional steps to update the variables on a multi–dimensional grid.

The basic one–dimensional time advancing step of the variables U_i at grid location i can be written in conservation form as

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta x}\left(F\left(U_{i+\frac{1}{2}}^{n+\frac{1}{2}}\right) - F\left(U_{i-\frac{1}{2}}^{n+\frac{1}{2}}\right)\right) \quad (6)$$

where $U_{i+\frac{1}{2}}^{n+\frac{1}{2}} = R(U_{i-3}, ..., U_i, ..., U_{i+4})$ is the solution of the Riemann problem between a left (U_L) and a right (U_R) state at the cell edge $i + \frac{1}{2}$. In PPM these states are constructed

by interpolation using information from 4 neighboring zones on each side of the cell edge $i + \frac{1}{2}$ (See Colella and Woodward 1984 for details). Therefore, the final time advance operator produced by PPM can be thought of as an operator with a 17-point stencil in two–dimensions, and a 25–points stencil in three dimensions, and the computational domain must include four zones of ghost points along each boundary. The integration time step Δt must be chosen to satisfy the Courant–Lewy–Friedrichs (CFL) condition $\Delta t = Crnm\ \Delta x/max|u + c_s|$ where c_s is the sound speed. A Courant number $Crnm$ of up to 0.9 can be chosen for the scheme to remain stable and robust.

The shock capturing properties that make Godunov methods robust come at the cost of great computational cost, as measured by the large number of operations required to update one grid zone (3000Flop). This property, combined with the locality of the data exchange patterns, implies that such algorithms scale very well on parallel machines, and that performance is ultimately limited by single node performance.

3.2. Nonlinear Multigrid Scheme

Multigrid methods were first introduced by Brandt (1977) to solve efficiently elliptic finite difference equations with typically O(N) operations on a grid with N zones. The basic idea is to accelerate the convergence speed of the basic relaxation iteration (e.g. a Jacoby or Gauss-Seidel iteration) by computing corrections to the solution on coarser grids (see e.g. Press *et al.* 1992 for a concise introduction to multigrid methods). The idea is that the low frequency components of the errors can be smoothed out more rapidly on the coarser grids.

In the V–cycle we use, one starts from the original grid where the problem is defined, or the finest grid. A few relaxation iterations are carried out on the finest grid to smooth the high frequency errors (*smoothing*), then the residual errors are injected into a coarser grid, typically with half grid points and twice the spacing (*injection*). This process of *smoothing* and *injection* continues down to a lowest grid, the coarsest grid, which can contain as few as one or four points. Here the correction to the solution can be found rapidly and can be interpolated back up to the finest grid through a succession of intermediate *smoothing* and *prolongation* steps. In figure 1 we show the fine and coarse grids that we use at each level. *Injection* and *prolongation* are obtained simply by bilinear interpolation. The particular choice for the distribution of points in the coarse grid is such that each grid maintains a cell–centered configuration with the physical boundaries being always located along the cell edges (i.e. staggered with respect to the grid points).

For the nonlinear scheme (4) and (5) the Full Approximation Storage (FAS) algorithm (see Press *et al.* 1992) is used. The relaxation step is accomplished using a time–lagged Gauss–Seidel iteration, whereby the nonlinear part of the operator (5), the κ coefficient, is evaluated at the the beginning of the iteration, time time step n.

4. Parallel Implementation: Domain Decomposition

We integrate the equations on rectangular domains. Therefore, data can be decomposed uniformly by assigning rectangular subdomains to each processor. Each subdomain must be augmented with a number of boundary cells (*ghost points*) necessary to compute the finite difference operators.

In general, finite difference operators can be thought of as having a stencil which deter-

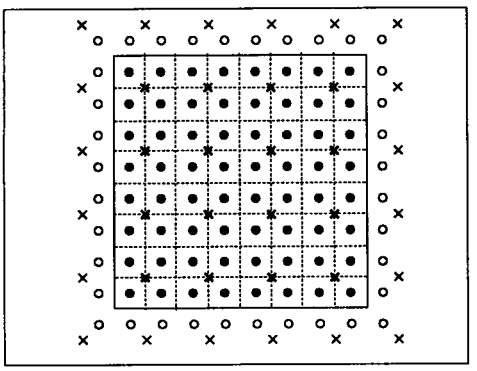


Figure 1. Grid hierarchy: • fine grid. × coarse grid.

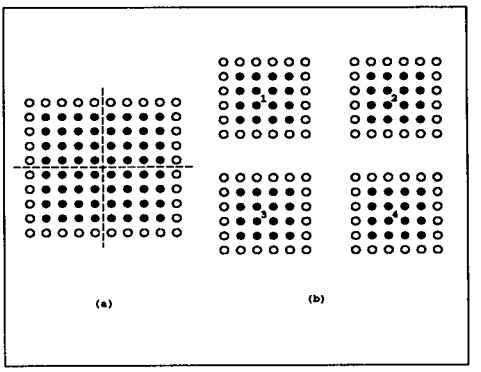


Figure 2. Domain decomposition on four processors.

mines the domain of dependency of one grid point from its nearest neighbors. The number of nearest neighbor points required depends on order of accuracy of the scheme, and it is 4 points in each direction for the PPM scheme and 1 point for the Crank–Nicholson scheme. Thus, the arrays on a $Nx \times Ny$ grid have $(Nx + 2g) \times (Ny + 2g)$ points, where g is the number of points in the computational stencil along each direction. Figure 2 shows an example of domain decomposition for our scheme. Because we use a cell–centered formulation, no interior points are duplicated across processors. However, quantities defined at the cell boundaries, like the fluxes in equation 6, are computed redundantly on neighboring processors to avoid the higher cost incurred in exchanging lots of data between processors. Obviously, this technique is likely to produce minimal degradation of parallel efficiency as long as the ratio N/p of grid points/number of processors is large.

The practical implementation of data decomposition and data exchange patterns between processors has been achieved using the tools available in the Blockcomm subset of PETSc.

The Godunov method, is ideally suited for parallel machines with medium grain parallelism. The algorithm involves extensive computations in each time step. Internode communication takes place in two forms. One is the local nearest neighbor exchange to update the ghost point. And the second is a global maximum to determine the CFL condition for the time step. Because the scheme uses the directional splitting, ghost points need to be updated twice in every time step, once after sweeping in the x direction and then after sweeping the y direction.

The multigrid algorithm has two different sections. The first section is similar to the Godunov algorithm. The physical grid is decomposed into subdomains, which are then mapped onto the processors. Each subdomain has its own grid hierarchy associated with it, down to a coarsest (but still distributed) grid that has only 2 points per direction on the processor. The ghost points in this algorithm need to be updated for each relaxation step, at all levels of the grid. As a result, the domain decomposition information must be

maintained for each grid level.

The second section begins when the coarsest mesh on the processor is reached. At this point, the communication cost for updating of ghost points becomes very expensive relative to the computation at each node. Besides, grid levels below this one would contain fewer than one point per processor, and one would then have to maintain considerable bookkeeping to keep track of grids distributed over a subset of processors. We have adopted the much simpler solution to collect all the subdomains of the coarsest grid globally onto each processor. Hence we continue with the process of injecting into coarser grids until we have a small enough grid to be solved exactly. The entire process of injecting, relaxing and projecting back is performed locally, and redundantly, on all processors. The redundancy in computations is offset by not having to broadcast the low-level corrections back to all processors.

The multigrid algorithm has a much lower computation to communication ratio compared to an explicit finite difference scheme because the amount of computation reduces much faster for each coarser grid than the amount of data exchanged, yet its data exchange patterns retain a large portion of locality. The only global data exchange required is performed on a small grid, and therefore the amount of data to be transmitted is small. Is is easy to see that this strategy will cause little parallel efficiency degradation as long as the ratio N/p is sufficiently large.

5. Experimental Results

In order to determine the scalability and performance properties of the code, we have performed extensive measurements and detailed performance analysis. Here we report the results the IBM SP-2 at the Maui Center for High Performance Computing. Similar experiments on the Intel Delta at Caltech and the IBM SP-1 at Argonne National Laboratory have brought essentially similar results. We have run a fixed number of time steps for problems with different grid sizes, and for each grid size we have varied the number of allocated processors.

The code has been instrumented with monitoring tools available within the Chameleon system. These tools allow a trace file for each run to be collected for post-mortem analysis. The analysis has been carried out using the MEDEA (REF) system, a tool which analyzes and summarizes information from the trace files. From the analysis we could determine, for example, the communication patterns of the algorithms, the length, duration, and distribution of the exchanged messages, and the amount of synchronization among processors. The complete performance analysis will be presented elsewhere (Tessera *et al.* 1995). In Fig. 3 we show the execution time curves.

As it can be expected, for each grid size the performance scales almost linearly until the *local* problem size on each processor becomes too small, in which case communications dominate over computations. At this point, an increase in the number of allocated processors does not produce a decrease in the execution times (in fact, it produces an increase), because of limits in bandwidth and latency. The effect is particularly strong in the multigrid part of the scheme, where the communication activities are most frequent.

However, this is not a real problem for us, since we are interested in studying problems on very large grids (in the range of 512^3), where close-to-linear scalability persists up

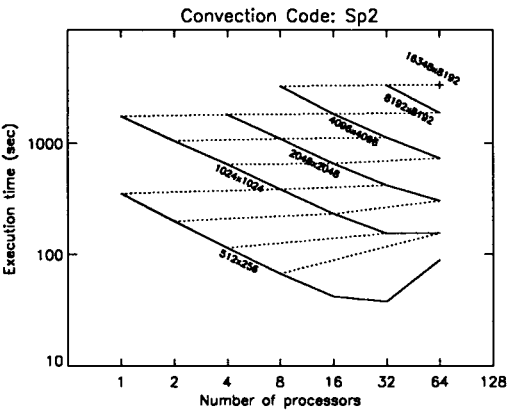


Figure 3. Execution time curve. The *solid* lines connect runs with the same *global* problem size. The *dashed* lines connect runs with the same *local* size on each processor.

to large numbers of allocated processors. In this regime, the code performance is mostly limited by the sustained computational performance on each node, and this is in turn limited by the large number of divides and sqrt() operations required in the Godunov algorithm. We estimate that the code achieves approximately 2.8 Gflops sustained performance on 64 IBM SP-2 processors (by comparison, the code runs at an average of 180 Mflops/Processor on a Cray YMP).

REFERENCES

1. Bogdan,T.J., Cattaneo, F., & Malagoli, A. 1993, Apj, 407, 316
2. Brandt, A. 1977, Math. Comput., 31, 333
3. Cattaneo, F., Brummell, N. H., Toomre, J., Malagoli, A., & Hurlburt, N. E. 1991, ApJ, 370, 282
4. Colella, P., & Woodward, P. 1984, JCP, 54, 174
5. Gropp, W. D., and Smith, B. 1993, "Users Manual for the Chameleon Parallel Programming Tools," Technical Report ANL-93/23, Argonne National Laboratory
6. W. Gropp and B. Smith, "Scalable, extensible, and portable numerical libraries, Proceedings of the Scalable Parallel Libraries Conference," IEEE 1994, pp. 87-93.
7. Press, W. H., Flannery, B.P., Teukolsky, S.A., & Vetterling, W.T. 1987, "Numerical Recipes", Ed. Cambridge University Press (Cambridge, UK)
8. Tessera, D., Malagoli, A., & Calzarossa, M. 1995, in preparation.
9. Calzarossa, M., Massari, L., Merlo, A., Pantano, M., & Tessera, D. 1995, IEEE Parallel and Distributed Technology, vol. 2, n. 4

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Towards Modular CFD using the CERFACS Parallel Utilities

Rémi CHOQUET and Fabio GUERINONI [a]
Michael RUDGYARD [b]

[a]IRISA/INRIA Campus de Beaulieu
35042 Rennes CEDEX, FRANCE

[b]CERFACS, 52 Av. Gustave Coriolis
31057 Toulouse CEDEX, FRANCE

We describe a modular, parallel computational fluid dynamics code. This is tested on dedicated workstations and scalable multiprocessors. Performances and comparisons for 2D and large 3D problems are given. We also present a preliminary description of a parallel implicit library.

1. Introduction: The CERFACS Parallel Utilities Library (CPULib)

The purpose of this paper is to report on recent progress in the development of a complex code for "modular" parallel fluid dynamics. The project was funded by the European Commission and involved several research centers, coordinated by CERFACS in Toulouse, France.

The starting point for the project was a large computational fluid dynamics code and its *generic* parallel library which has been called the *CERFACS Parallel Utilities* (CPULib) [9,8].

In current CFD terminology, it can be described as a parallel unstructured/structured, two and three dimensional Navier-Stokes solver. Its supporting parallel routines (CPULib) consist of message–passing implementations of generic utilities that are required by many numerical schemes in computational fluid dynamics. The library is built upon the IPM macros [5], which allow us to use message passing software such as PVM, PARMACS and MPI. Recent work has been oriented toward the achievement of two main goals:

The first goal was to test the performance of parallel code, not only on networked multiprocessing (with several hosts running the same code), but also on truly scalable distributed memory multiprocessors like the Intel Paragon. The message passing library of choice in this case has been PVM. Although PVM was conceived for parallel processing on networks of heterogeneous workstations, there are few large-scale codes which run and perform well on true scalable multiprocessors.

The second goal was the definition of a suitable approach for the implementation of implicit methods within the farmework of CPULib. We describe below the methods and routines already incorporated.

The paper is organized as follows. Following this introduction, we give a detailed

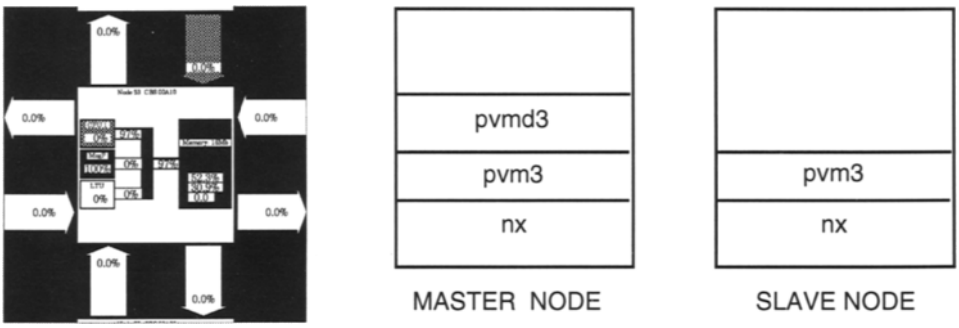


Figure 1. Paragon node (left) and message passing layers (right)

description of the multiprocessor on which part of the code was tested. The following sections cover the numerical schemes and the presentation of test problems. To close, we sketch the structure of the implicit library and its interaction with CPULib.

2. The Intel Paragon Multiprocessor

In the previous section we have seen how CPULib is completely independent of the message passing system used by the underlying processors of our parallel machine. Due to its popularity and flexibility, results shown in this paper are computed using PVM. Although this is not the native language of the Intel Paragon, we shall see that an acceptable performance is obtained.

2.1. Hardware description

At IRISA, we have a small scale Paragon, Model XP/S A with a total of 56 user nodes, 3 for service, and 2 for I/O. Each nodes has two i860 XP with 16Mb of memory. They are rated at 75Mhz, 75MFLOPs (peak d.p.) and include a small 16Kb cache.

The grid network with no wraparound has a theoretical speed of 200Mb/s (for reference, the bandwidth of the cache-memory is 400Mb/s). Although the network has been rated at 165Mb/s under the SUNMOS operating system, the bandwidth for a Model XP/S A4E with nodes of 16Mb, and running OSF/1, has been rated at only 48Mb/s for FORTRAN code using NX [7] — we expect this to be even less under PVM.

2.2. PVM on multiprocessors

The PVM (Parallel Virtual Machine) system was developed to permit an heteregeneous cluster of workstations to work together in the solution of some scientific problem. Moreover, tasks assigned to each processor can be made according to the capability of each host.

In view of limitations associated with shared memory processors, the need to develop a multiprocessor (MP) version of PVM was evident. Such capabilities were incorporated

for the first time in its version 3.2 in late 1993 [6,4].

In spite of the success of PVM in a multihost environment, it has, until recently, been generally ackowledged that PVM on a MP is still relatively fragile. Indeed, a number of restriction apply on the Paragon:

- *One task per node* — we must have at least as many computational nodes as there are tasks, and must prevent forking on a node.
- *Only one pvmd (daemon)*, which must be constantly probing the MP tasks (Figure 1). Inter-host communications can be very slow.
- *No stderr/stdout* from the nodes since there is no */tmp/pvmdl.user* which exist on normal hosts.

3. Explicit Schemes

The convective part of the code is based on a cell-vertex finite volume discretization, and as such can also be viewed as finite difference schemes [9]. The explicit schemes are normally very robust and will be described first, together with results.

3.1. The Numerical Scheme

We solve the 3-D Navier-Stokes equations in conservation form:

$$\mathbf{u}_t + \nabla \cdot \mathcal{F} = 0 \tag{1}$$

where $\mathcal{F}$ involves convection and diffusive parts. The explicit step of our algorithm is fundamental, since it is also used as the predictor step in the implicit formulation. We use a (cell–vertex) finite volume version of the Lax-Wendroff scheme. In one dimension, this can be written

$$\Delta w_{expl} = -\Delta t \mathbf{F}_x^n + \frac{\Delta t^2}{2}(A^n \mathbf{F}_x^n)_x \tag{2}$$

It is well–known that such a scheme is second order in time and space and is stable if the Courant number (CFL) is less than one; this poses a fundamental restriction, as with any explicit scheme. It is also well–known that when presented with discontinuous initial values, the scheme produces spurious oscillations. At the expense of more computation, implicit schemes try to improve upon explicit schemes in both senses.

3.2. Performance on Networked Workstations

For preliminary tests, we have chosen two test cases using small meshes and whose characteristics are described in Table 1.

The original unstructured meshes are decomposed (by routines incorporated in CPULib) into *blocks*, each of which is assigned to a specific host (or node, in the case of a multiprocessor). Computations are performed globally using message passing routines for sending information across blocks. A detailed discussion is given in [8].

The meshes represent standard supersonic NACA 0012 problems, with no viscosity. Since explicit methods do not involve the use of iterative methods for solving any linear

Table 1
Test Meshes

Name	Partitions	Nodes	Cells	Bound. Nodes	Shared Nodes
1bl	1	2104	4047	165	0
4bl	4	2303	4047	175	402
8bl	8	2399	4047	175	612

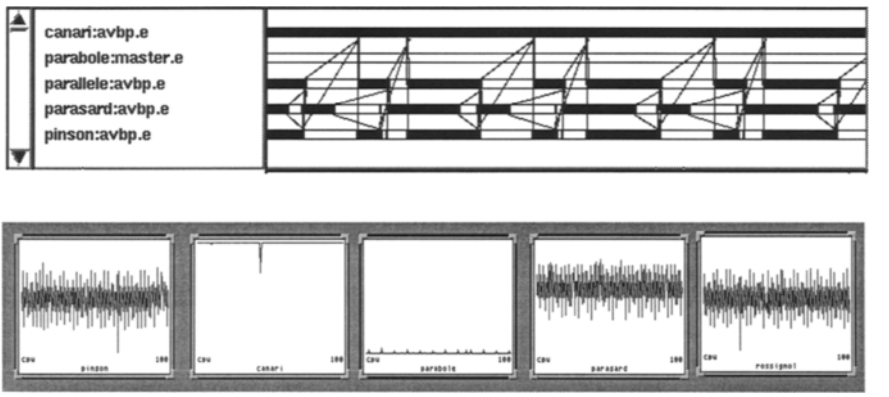


Figure 2. CPU utilization with *perfmeter* (left) and space-time diagram with *xpvm*

systems, the present tests do not assess any *global* elementary operations such as norms, scalar products, and matrix vector products.

Basically, all meshes are the same except for the number of blocks. The last two columns indicate the number of nodes on the physical boundary, and nodes which are at the interface of blocks.

All computations were evaluated on up to 9 dedicated SPARC Sun workstations. In preliminary computations we obtained low efficiencies (for example a speed-up of only 3.65 with 8 processors). Due to the nature of the code, which is nearly all parallel, this poor performance could not be explained in terms of communication overhead alone. However, using tools like *xpvm* and SUN *perfmeter*, it was easy to discover that one of the hosts ran at a peak of less than $10 Mflops$ (host *canari* in Figure 3.2) while the others had peaks of $21 Mflops$ (SPECfp92) at least. By removing the bottleneck we obtain efficiencies of nearly 0.82.

3.3. Performance on the Paragon

We repeated the tests made for the networked suns with 2, 5, and 9 nodes of the Paragon. The master is not active in the computation and this must sit on a node by itself with no other process. As explained earlier, this is an intrinsic restriction of PVM

Table 2
Peformance on Networked Workstation

Mesh	SUN4 Proc*	Time/iteration	Wall-time	Speed-up
1bl	2	7.96	3184	1.0
4bl	5	2.06	824	3.86
8bl	9	1.22	488	6.52

on a multiprocessor.

In the worst case (9 nodes) the efficiency was nearly 0.95, with a run–time improvement of about 12 times that of the corresponding computations on network Sun's at $22Mflops$. This is much better than what could be explained in terms of raw processor power. Space limitations prevent us from saying more about such computations, and we turn instead to more realistic problems.

Three-dimensional Computations

Table 3 describes two tetrahedral meshes for an ONERA M6 wing. The problem is a transonic Euler computation at $M = 0.84$, $\alpha = 3.06°$.. Partitioning of the meshes was done automatically using *CPULib* routines. The right part of the table indicate the minimum and maximum number of nodes of the resulting blocks.

Table 3
Three Dimensional Meshes and Block Sizes

Mesh	Nodes	Cells	Partit.	Max Size	Min Size
m6_co	10,812	57,564	1	10,812	10,812
			2	6,126	6,089
			4	3,548	3,207
			8	2,065	1,790
			15	1,322	988
m6_fi	57,041	306,843	1	57,041	57,041
			2	31,949	31,063
			4	17,586	16,067
			8	9,484	8,031
			15	5,616	4,497

The computations shown in the table are for the fine mesh *m6_fi* with more than 300,000 cells. Note that there are two columns: *S.up*1 and *S.up*2. The first gives the conventional speed-up – the relative speed when compared to the execution with only one processor. This leads to values much larger than the number of processors.

This effect is clearly due to swapping overheads, since for less than four processors, the partitions (blocks) are two small to fit comfortably on the $16Mb$ of real memory, and virtual memory mechanisms start working, resulting in inflated execution times. For this reason we use the second figure *S.up*2 — speed-up with respect to 8 procesors. Even so, the calculation with 15 blocks still gives anomalous speed-ups. This is likely to be due to

cache and page misses because of indirect addressing.

Finally, results are shown for the same problems with dedicated SUNs. Notice that swapping is still considerable on a Paragon node (16Mb) using 4 blocks, so that the slower but larger (32Mb-48Mb) SUN's take the lead. With more memory on the nodes, we expect the Paragon to be faster by at least the factor by which its processor is more powerful than that of the SPARC's.

Table 4
Three Dimensional Meshes and Partition Sizes

Partit.	Time/iteration	S.up1	S.up2	Time/It. SUN
1	240.5	1.0	-	-
2	175.2	1.37	-	-
4	74.8	3.215	-	61.8
8	8.2	29.329	1.0	33.6
15	2.8	85.89	2.92	-

4. Toward an implicit library for CPULib

The step size of the time integration of (2) is limited by stability. As a result, we would like to consider implicit schemes, even though the solution of such schemes on unstructured meshes is non-trivial. The goal of this part of the work is to define the structure of an easy-to-use library of algorithmic modules for parallel processing. Since this structure must be applicable to several different application codes, it must keep all algorithmic and problem specific data separate. As stated before, the framework for this analysis is a fluid dynamics code and its generic parallel library CPULib.

4.1. The Implicit Equations

We choose to implement an extension of the Lax-Wendroff scheme due to Lerat[2]. We first consider linear implicit schemes. In one dimension these take the form:

$$\Delta w - \frac{\Delta t^2}{2}[(A^n)^2(\Delta w)_x]_x = \Delta w_{expl} \tag{3}$$

where w_{expl} is the predictor term and is defined by (2). For the nonlinear version one solves

$$\Delta w - \frac{\Delta t^2}{2}[(A^{n+1})^2(\Delta w)_x]_x = -\Delta t \mathbf{F}_x^n + \frac{\Delta t^2}{2}(A^{n+1}\mathbf{F}_x^n)_x \tag{4}$$

Both implicit schemes are linearly stable for any time step, are second order accurate in time and do not require artificial viscosity. Furthermore, the second method has proven useful for unsteady flows with shocks[3].

4.2. The implicit library

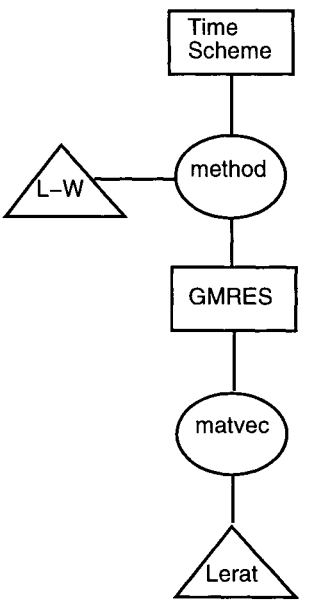
In the context of CPUlib, the definition of a modular framework should allow several application codes to make use of the library. The structure of the library should also

imply minimal work for the user. As a result, we have precluded the use of common blocks between library modules and the user's own routines. This constraint implies the creation of low-level routines which can be used within the high-level communication routines of the implicit library.

Typically, the library consists of three components:

- a 'black–box' *module* which makes use of *CPUlib*'s data–structure, as well as low–level parallel routines ;
- an *interface* routine, required by the library although supplied by the user, that links the modules and *problem specific routines (PSR)* ;
- *problem specific routines* (optional).

In the diagram exemplifying the use of the library, modules are denoted by *rectangles*, *PSR* by *triangles* and interfaces by *ovals*. As an example, consider one time step of the linear implicit Lerat scheme, illustrated on the right. Once we have defined the Lax-Wendroff residual Δw_{exp}, the solution of (3) involves the following three components:



- We call the *GMRES* module to solve the linear problem (3) ;
- the *GMRES* routine calls the interface routine *matvec* ;
- the routine *matvec* calls the *Lerat* residual.

For the non linear scheme (4), one additional level is necessary. In this case, the *Newton* module calls an interface module which calls the module *GMRES*. Here *GMRES* involves Jacobian vector products and matrix-free techniques.

5. Final Remarks

The basic tenet of the original code, which is its capability to adapt to different message passing parallel systems, has been demonstrated. We have considered a very powerful class of machines, (which in theory may consist of thousands of processors) using the popular message–passing software PVM. The combination of portable software as well as these powerful machines should open the way for large–scale parallel computations for fluid dynamics problems.

There are, however, stumbling blocks. Implicit computations using unstructured meshes have as yet proven difficult to implement due to the complexities in treating the boundaries implicitly, even in the steady case. Simple experiments have shown that such boundary conditions should not be treated as natural extensions of the explicit scheme, since this can result in spurious solutions, and bad (or no) convergence.

REFERENCES

1. *Paragon Fortran Compiler's User's Guide.* Intel Corporation, on: 312491-002 edition, March 1994.
2. A. Lerat and J. Sides. – Efficient solution for the steady Euler equations with a centered implicit method. – In K.W. Morton and M.J. Baines, editors, *Numerical Fluid Dynamics III*, pages 65–86, Oxford, 1988. ONERA, Clarenton Press.
3. Anne-Sophie Sens. – Calcul d'écoulements transsoniques instationnaires par résolution implicit centrée des équations d'Euler sans viscosité artificielle. – Technical Report 1990-8, ONERA, 1990. – Thèse doctorat, Univ. Paris VI.
4. A.T. Beguelin, J.J. Dongarra, and *et al. PVM version 3.3 (Readme.mp).* U. of Tennessee, ORNL, Emory U., edition, Nov 1994.
5. The FunParTools Co. *IPM2.2 User's Guide and Reference Manual.* CERFACS, edition, September 1994.
6. A. Geist and A. *et al* Beguelin. *PVM: Parallel Virtual Machine.* Volume of *Scientific and Engineering Computation*, The MIT Press, edition, 1994.
7. Ph. Michallon. *Etude des Perfomances du Paragon de l'IRISA.* , IRISA, 1994.
8. M. Rudgyard and T. Schönfeld. CPULib – a software library for parallel applications on arbitrary meshes. 1995. To be presented at CFD'95, Pasadena,CA.
9. M. Rudgyard, T. Schönfeld, R. Struijs, and G. Audemar. A modular approach for computational fluid dynamics. In *Proceedings of the 2nd ECCOMAS-Conference*, Stuttgart, September 1994.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A fully implicit parallel solver for viscous flows; numerical tests on high performance machines

E. Bucchignani[a] and F. Stella[b]

[a]C.I.R.A. , Capua (CE), Italy

[b]Dip. di Meccanica ed Aeronautica, Univ. di Roma "La Sapienza", Italy

In this work a parallel fully implicit flow solver for incompressible Navier-Stokes equations, written in terms of vorticity and velocity, is presented and tested on different architectures. Governing equations are discretized by using a finite difference approach. Linear systems arising from discretization are solved by a parallel implementation of Bi-CGSTAB algorithm used in conjunction with a Block Incomplete LU factorization as preconditioner. Results are given for the simulation of a 3D flow in a rectangular box heated from below. Emphasis is given in the analysis of transition from steady to non periodic regime.

1. INTRODUCTION

The continuous progress in the development of solution codes for unsteady, three dimensional fluid dynamics problems is unthinkable without the substantial increase of computational speed and storage capacity of high performance computers. A change from vector supercomputers to parallel systems is currently taking place, therefore the need of parallel software that is portable, load balanced and scalable to hundreds of processors is often felt.

Most of the CFD packages used in practical applications are based on Domain Decomposition techniques [1,2], that allow to obtain very high parallel performances. However, they are not suitable if an unsteady flow has to be studied. In this case it is necessary to use a fully implicit approach, usually preferred over the explicit one also for steady state calculations, because of its good numerical stability, especially when used with large time steps. Due to the large amount of inter-process communication required from the implicit methods, it is very hard to obtain a parallel speed-up on distributed parallel machines. Nevertheless, the parallelization of these methods is a current research area and has been investigated by many authors [3 7] and obtained results are encouraging. In this paper we present numerical results related to a parallel implicit solver for viscous flows, based on the vorticity velocity formulation [8] of Navier-Stokes equations. A fully implicit approach has been adopted in order to tackle this problem, leading to a large sparse linear system per each time step. Bi-CGSTAB [9] has been proved to be very cost-effective for the solution of this type of problems. Performances, in terms of parallel speed-up, are strongly affected by the choice of an effective parallel preconditioner. As a good compromise between the efficiency of ILU and the facility of implementation of

Diagonal scaling, a Block Incomplete LU factorization (BILU) has been chosen.
The application considered is a three-dimensional flow in a rectangular box, bounded by rigid impermeable walls and heated from below (i.e. Rayleigh - Benard problem). Vertical walls are adiabatic, while horizontal surfaces are assumed to be isothermal and held at different values of temperature. Numerical results are presented for several values of the characteristic parameters and compared with those obtained in the sequential case. Parallel performances are presented in terms of elapsed time and parallel speed-up as a function of the number of processors.
Parallel code has been developed by using PVM 3.3 environment, as *message passing* software. It was implemented in double precision on different machines, such as Convex Metaseries, Convex Exemplar, Cray T3D and IBM-SP2.

2. THE MATHEMATICAL MODEL

The vorticity velocity form of the non-dimensional equations governing three-dimensional natural convection in a Newtonian fluid, assuming the Boussinesq approximation to be valid, is:

$$\frac{1}{Pr}\frac{\partial \omega}{\partial t} + \frac{1}{Pr}\nabla \cdot (\mathbf{u}\omega) = \nabla^2 \omega - Ra \nabla \times \left(\theta \frac{\mathbf{g}}{|\mathbf{g}|}\right) \tag{1}$$

$$\nabla^2 \mathbf{u} = -\nabla \times \omega \tag{2}$$

$$\frac{\partial \theta}{\partial t} + (\mathbf{u} \cdot \nabla)\theta = \nabla^2 \theta \ . \tag{3}$$

The standard conservative form is adopted for the convective terms in the vorticity transport equation in order to conserve exactly the mean vorticity as prescribed by Stokes theorem. The non-dimensional scheme adopted is defined in [10] as *Conduction eq. convection*. It uses the non dimensional parameters $Ra = g\beta\Delta T l^3/\kappa\nu$ and $Pr = \nu/\kappa$.
The mathematical formulation is completed by proper boundary conditions, that are simple to enforce and implement [11].
The governing differential equations are discretized on a regular mesh by using the finite difference technique. Spatial derivatives are discretized using second order central differences, while time derivatives are discretized using a three point second order backward difference. Discretization gives rise to a linear system of equations of type $\mathbf{A}x = b$ for each time step.

3. DESCRIPTION OF ALGORITHM

3.1. Iterative method

As discussed in the previous section, a large sparse linear system has to be solved to advance the solution in time at each time step. In this work a parallel implementation of Bi-CGSTAB algorithm (iterative method of the Conjugate Gradient class) has been used; it has been proved to be very efficient and stable, even if coefficient matrix $\mathbf{A}$ is non symmetric. Bi-CGSTAB is widely described in [9] and is not reported here for seek of brevity. On the contrary, it is useful to describe here some details used in order to

obtain an efficient implementation of the method. Let n be the number of equations and *nproc* the number of processors used. Basic operations of the method are: vector update, matrix-vector products and inner products.
Vector update can be executed in a easy way simply splitting the vector components among processors. Communications are not required and a linear speed-up is expected for this kind of operation.
Matrix-vector product ($\mathbf{A}t$) is executed by splitting the matrix into horizontal bands and mapping them among processors. Some components of t vector have to be transferred among processors: a very effective procedure has been adopted, by transferring to each processor only the components really needed. If non-zero elements of $\mathbf{A}$ are enclosed in a narrow band, only neighbour communications are required, so communication time does not increase for increasing values of *nproc*.
Inner product requires global communications for assembly the sum and for distribution of the assembled sum over the processors. Implementation of inner products represents a very tough problem because of the small granularity of each computational task. As matter of fact, it limits the number of processors for which the maximum efficiency is obtainable. Global time for execution of inner product is given by:

$$T = t_{calc} + t_{comm} = K_{calc}\frac{n}{nproc} + K_{com} * nproc \tag{4}$$

where K_{calc} and K_{com} are constant quantities depending on the characteristics of the machine under test, but it is clear that *nproc* cannot be too large.

3.2. Preconditioner

In order to fulfill stability and convergence of the iterative solver, the use of a preconditioning technique is essential. It consists of finding a real matrix C such that the new linear system

$$C^{-1}\mathbf{A}x = C^{-1}b \tag{5}$$

has (by design) better convergence and stability characteristics than the original system. Several preconditioners have been developed: Diagonal Scaling and ILU factorization are two of the most widely used. In the first case C is diagonal, whose elements are the same as the main diagonal of $\mathbf{A}$. In the second case C is defined as the product LU of a lower (L) and an upper (U) triangular matrix, generated by a variant of the Crout factorization algorithm: only the elements of $\mathbf{A}$ that are originally non-zero are factorized and stored. In this way the sparsity structure of $\mathbf{A}$ is completely preserved. As matter of fact, parallelization of diagonal scaling is straightforward, but it does not allow us to significantly reduce the iteration number required for convergence; on the other hand, ILU preconditioner is very good on sequential computers, but it cannot be efficiently implemented on a distributed memory machine. In order to overcome these problems, the BILU preconditioner has been proposed in [12]. This preconditioner is based on a block decomposition of matrix $\mathbf{A}$, with no overlapping among the diagonal blocks: each processor performs an ILU factorization only on a square block centered on the main diagonal. This preconditioner does not affect the final result obtained by using Bi-CGSTAB, even if it could affect the number of iterations required (see Tables in [12]).

4. THE PHYSICAL PROBLEM

The numerical code described above has been used to study a three-dimensional flow in a rectangular box, bounded by rigid impermeable walls and heated from below (Rayleigh - Benard problem). Vertical walls are adiabatic, while horizontal surfaces are assumed to be isothermal and held at different values of temperature. The Rayleigh-Benard problem has been studied over the past century by a large number of researchers, because it has many practical applications, as well as theoretical interests (transition to turbulent regime) [13–16]. Gollub & Benson experimentally showed that there are several routes that conduct to turbulent convection (in small boxes). The general trend reported was that an increase in the Rayleigh number leads to more and more complex behaviour, via discrete transitions. In the present work a system characterized by 2.4 * 1 * 1.2 geometry, filled with water at 33^0C ($Pr = 5.$) is adopted as study case. The Rayleigh number has been varied from 40000 to 180000, in order to investigate the behaviour of the system from steady state to non periodic regime. A mesh with $25 * 21 * 25$ points and a time step $\Delta t = 10^{-4}$ have been used for numerical discretization, that revealed themselves accurate enough for our purposes. Only to evaluate parallel performances, another test case has also been considered: it involves a computational domain with $1.5 * 1 * 4.5$ dimensions, assuming $Pr = 400$ and $Ra = 70000$ and adopting two meshes: $16 * 21 * 46$ and $32 * 21 * 46$.

5. PARALLEL ENVIRONMENT DESCRIPTION

The numerical simulations have been executed on the the following machines:
- CONVEX METASERIES, installed at CIRA, is a cluster of 8 HP PA-735 workstations, connected by FDDI ring. The peak performance of each processor is 200 MFLOPS at 64 bit and each node has 64 MB of local memory.
- CONVEX EXEMPLAR, installed at CIRA, is comprised of 2 hypernodes, each of which has 8 HP PA-RISC 7100 chips, an I/O port, and up to 2 gigabytes of physical memory on a non-blocking crossbar sustaining a bandwidth of 1.25 gigabytes/second. Hypernodes are interconnected with 4 rings, each with raw bandwidth of 2.4 gigabytes/second. Sustainable bandwidth between any pair of nodes is 160 megabytes/second in each direction, up to 8 pairs active simultaneously. Exemplar can be programmed as a distributed memory message-passing machine (by using PVM or MPI) or as a conventional shared memory machine; in fact both the C and Fortran compilers can automatically parallelize data-independent loops. In this work we have programmed it as a distributed memory machine.
- CRAY T3D, installed at CINECA (Bologna, Italy), is a distributed memory parallel system with 64 processing nodes. The processing nodes are DEC Alpha processors and are connected by a 3D bidirectional torus network. Each switch of the network is shared by two nodes. The memory is physically distributed, but globally addressable, so that both SIMD and MIMD programming models are supported. A Cray C90 vector processor is used as host system for the T3D architecture.
- IBM SP2, installed at CASPUR (Roma, Italy), is a distributed memory system with 8 "thin nodes" (based on POWER2 processor). The nodes are interconnected by a 'High-Performance Switch'. The switch is a multistage omega network that performs wormhole routing. An optimized version of PVM (named PVMe) is available.

6. RESULTS

6.1. Parallel results

The numerical code has been developed by using PVM 3.3 environment, as *message passing* software. Numerical tests have been executed on the meshes 25∗21∗25, 16∗21∗46 and 31∗21∗46, leading respectively to systems with 104104, 123046 and 231616 equations (test cases A, B, C). Parallel performances are presented in terms of elapsed time (seconds) and parallel speed-up as a function of the number of processors. Tab. 1, 2 and 3 show performances obtained on the various architectures. Besides, test cases A and B have been executed on a IBM RISC 355, on Convex 34 and Convex 38 (these last two have respectively 2 and 6 vectorial shared memory processors). Tab. 4 shows performances obtained on these machines, in terms of Cputime.

Table 1
Performances related to the the test case A (104104 eq.). $Pr = 5$, $Ra = 60000$, two time steps.

	Metaseries		Exemplar		T3D		SP2	
n. pr.	time	sp.-up	time	sp.-up	time	sp.-up	time	sp.-up
1	183	1.00	244	1.00	391	1.00	77	1.00
2	108	1.69	126	1.94	244	1.60	40	1.92
4	75	2.44	79	3.08	156	2.51	26	2.96
6	60	3.05	63	3.87	112	3.49	21	3.66
8	48	3.81	57	4.28	95	4.11	-	-

Table 2
Performances related to the the test case B (123046 eq.). $Pr = 400$, $Ra = 70000$, two time steps.

	Metaseries		Exemplar		T3D		SP2	
n. pr.	time	sp.-up	time	sp.-up	time	sp.-up	time	sp.-up
1	401	1.00	545	1.00	840	1.00	161	1.00
2	245	1.63	309	1.76	490	1.71	87	1.85
4	133	3.01	156	3.49	269	3.12	55	2.92
6	107	3.74	119	4.58	198	4.23	45	3.58
8	97	4.13	106	5.14	176	4.77	-	-

These results show a good behaviour of the method, being an efficiency of about 67% on 8 processors for the heaviest case. The inevitable loss of efficiency is especially due to the less effectiveness of the preconditioner BILU as the number of processors is increased. Besides, the inter-processor communication time is not negligible, although a suitable strategy has been adopted. The load unbalancement has also to be taken into account: in fact, especially for a coarse grid, it could cause a loss of efficiency of about 10% on 8 processors.

6.2. Fluid dynamics results

Numerical simulations have highlighted that for $Ra = 40000$ the flow is steady and organized as two symmetric counter-rotating rolls (Fig. 1 and 2). At $Ra = 50700$ the system

Table 3
Performances related to the the test case C (231616 eq.). $Pr = 400$, $Ra = 70000$, one time step.

	Metaseries		Exemplar		T3D		SP2	
n. pr.	time	sp.-up	time	sp.-up	time	sp.-up	time	sp.-up
1	360	1.00	474	1.00	805	1.00	333	1.00
2	223	1.61	258	1.84	435	1.85	168	1.98
4	125	2.88	144	3.29	234	3.43	99	3.36
6	99	3.63	110	4.31	178	4.51	67	4.97
8	79	4.55	82	5.64	148	5.42	-	-
12	-	-	64	7.40	110	7.32	-	-
16	-	-	53	8.94	92	8.75	-	-

Table 4
Performances of the sequential version of the code.

	IBM RISC	Convex 34	Convex 38
Case A	284	366	170
Case B	540	658	382

undergoes a first bifurcation to periodic regime, and at $Ra = 60000$ a FFT executed on the U_x velocity component transient history in a fixed point shows that the flow is periodic with a main frequency of 29.29 (and its harmonics)(Fig. 3-a). An increase of Ra up to 10^5 causes the the transition to quasi-periodic regime. In fact the power spectrum reveals the existence of two fundamental frequencies (39.06 and 80.56) and several harmonics given by linear combinations of the fundamental ones (Fig. 3-b).
At $Ra = 1.4 * 10^5$, a *phase locking* phenomenon happens: the two fundamental frequencies are modified in such a way that their ratio is a rational number (2, in this case); the flow is periodic, again (Fig. 3-c). At $Ra = 1.8 * 10^5$ a non periodic behaviour was found: the power spectrum is not regular at all and main frequencies are not appreciable. However, a deeper analysis of this flow seems to be necessary in order to understand the difficult problem of transition to *chaos*.

7. CONCLUSIONS

The aim of this study was to look at the feasibility of implementing a fully implicit solver for Navier-Stokes equations on distributed memory MIMD architectures. Performances obtained are encouraging: in fact we are able to investigate a tough problem, such as the transition to unsteady regime in a reasonable computational time.
It has been shown that the code can be ported to a range of parallel hardware, being a good parallel efficiency and scalability on all the machines considered. Worse performances in terms of elapsed time have been obtained on Cray T3D. This is mainly due to the less computational power of the node processor of this supercomputer.

Acknowledgements
The authors would like to thank:
- ing. F.Magugliani of CONVEX ITALIA (Milan) for the help supplied in the execution of numerical tests.
- CASPUR (c/o University of Rome) for providing computational resources in the executions of numerical tests.

REFERENCES

1. Schiano P., Matrone A., Proc. Parallel CFD '93, Paris, May 10-12, 1993.
2. Tromeur-Dervot D., Roux F., Proc. Parallel CFD '93, Paris, May 10-12, 1993.
3. Wake B., Egolf T., AIAA Journal, **29**(1), pp. 58-67, 1991.
4. Long L., Khan M., Sharp H., AIAA Journal, **29**(5), pp. 657-666, 1991.
5. Carriere P., Jeandel D., Int. J. Num. Meth. Fluids **12**, pp. 929-946, 1991.
6. Tsai M., Proc. Parallel CFD '93, Paris, May 10-12, 1993.
7. Schreck E., Peric M., Int. J. Num. Meth. Fluids **16**, pp. 303-327, 1993.
8. Guj G., Stella F., J. Comp. Physics, **106** (2) pp. 286-298, 1993.
9. Van der Vorst H., SIAM J. Sci. Stat. Comput, **13**, n. 2, pp. 631-644, 1992.
10. de Vahl Davis G., Heat Transfer 1986, **1**, pp. 101-109, Hemisphere Publ. Corp., Washington, 1986.
11. Stella F., Bucchignani E., "A fully implicit vorticity-velocity method using preconditioned Bi-CGSTAB", submitted to Int. J. Num. Meth. Fluids, 1995.
12. Stella F., Marrone M., Bucchignani E., Proc. Parallel CFD '93, Paris, May 10-12, 1993.
13. Davis S., J. Fluid Mech., **30**(3), pp. 465-478, 1967.
14. Clever R.M., Busse F.H., J. Fluid Mech, **65**, pp. 625-645, 1974.
15. Gollub J.P., Benson S.V., J.Fluid Mech. **100**, pp. 449-470, 1980.
16. Stella F., Guj G., Leonardi E., J.Fluid Mech, **254**, pp. 375-400, 1993.

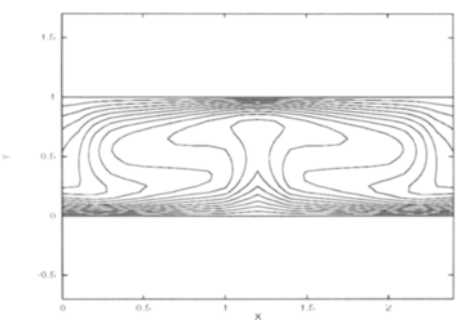


Figure 1. Contour plot of temperature in the plane $z = 0.6$ at $Ra = 40000$.

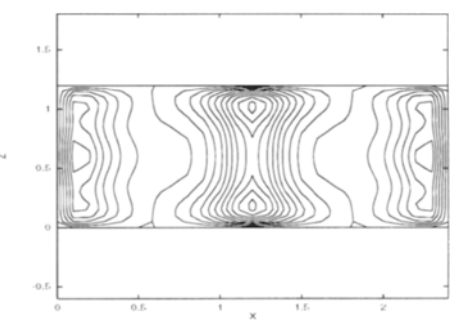


Figure 2. Contour plot of V-velocity in the plane $y = 0.5$ at $Ra = 40000$.

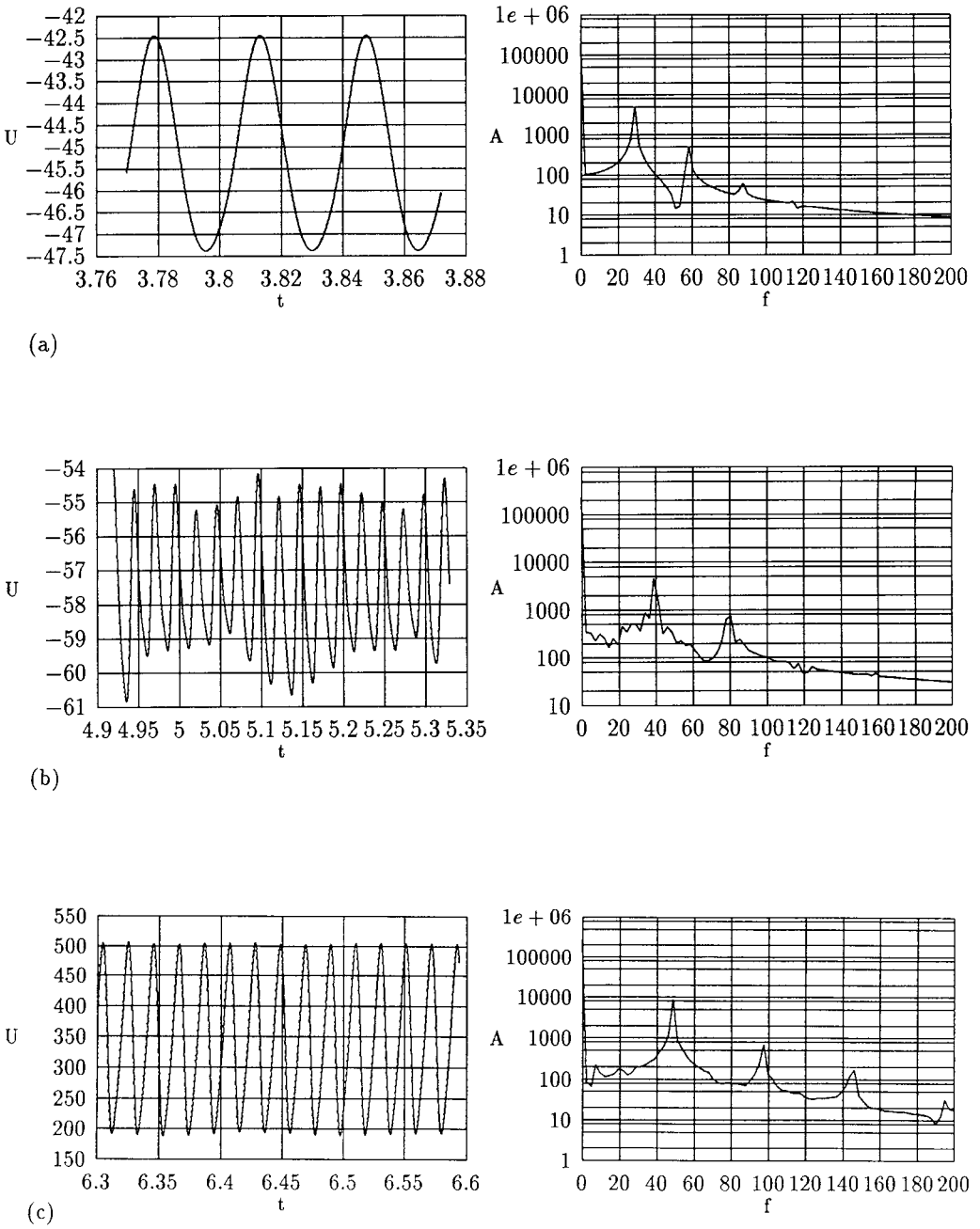


Figure 3. U-velocity as a function of time and power spectrum for: (a) $Ra = 60000$; (b) $Ra = 100000$; (c) $Ra = 140000$.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Space- and Time-Parallel Navier-Stokes Solver for 3D Block-Adaptive Cartesian Grids

V. Seidl, M. Perić[a] and S. Schmidt[b]

[a]Institut für Schiffbau, Universität Hamburg, Lämmersieth 90, D-22305 Hamburg, Germany; e-mail: seidl@schiffbau.uni-hamburg.de

[b]Hermann-Föttinger-Institut, Technische Universität Berlin, Straße des 17. Juni 135, 10623 Berlin, Germany

We introduce a parallelization strategy for implicit solution methods for the Navier-Stokes equations based on domain decomposition in time and space and discuss in particular the use of parallel preconditioned conjugate gradient solvers. A fully conservative second order finite volume discretization scheme with non-matching block interfaces in a block-structured grid is described. The parallelization method is applied to a direct numerical simulation of turbulent channel flow over a square rib at Reynolds number $Re = 3.7 \cdot 10^3$, based on the mean velocity above the obstacle and the step height. Results are compared with data from literature and the efficiency of the method is analyzed.

1. Overview and Outline of Numerical Method

The discretization of the Navier-Stokes equations is based on a fully implicit, second order, three time levels scheme in time and a fully conservative, second order finite volume method in space using collocated variable arrangement and central differencing.

The equations are solved by a segregated iterative approach (SIMPLE method) with non-linearities and inter-equation coupling being resolved in a predictor-corrector scheme within outer iterations [1]. In the predictor step linearized momentum equations with pressure and mass fluxes of the previous outer iteration are used to determine intermediate velocities. To fulfill the constraint of continuity, a Poisson-type pressure-correction equation has to be solved in the corrector step. This sequence is repeated within each time step until all equations are satisfied (about ten outer iterations per time step in the present study). All linear systems are relaxed by ILU-based methods (inner iterations), which are accelerated by the conjugate gradient (CG) method in case of the symmetric pressure-correction equation.

Domain decomposition in space and time is introduced in the next section. The following section presents a parallel preconditioned conjugate gradient method and its numerical efficiency. The implementation of block structured grids with non-matching block interfaces in a parallel environment is described in the fourth section. Finally, the method is applied to a direct numerical simulation of low Reynolds number turbulent channel flow over a wall mounted rib.

2. Domain Decomposition in Space and Time

The concurrent algorithm is based on data parallelism in both space and time and has been implemented under the SPMD paradigm using message passing.

For space parallelism the solution domain is decomposed into non-overlapping subdomains, each subdomain being assigned to one processor [1]. A memory overlap of one control volume wide layer along interfaces has to be updated within inner iterations by a local communication between neighbour subdomains. In the SIMPLE method, the assembley of the coefficient matrices and source terms in outer iterations can be performed in parallel with no modification of the serial algorithm, as only variable values from the previous outer iteration are required. Therefore, with regard to decomposition in space, only the linear solver needs to be considered, which will be the subject of the next section.

For time dependent problems, the iterative nature of the presented implicit time scheme allows parallelism in time. This is useful for problems of moderate size, where domain decomposition in space gets less efficient with increasing number of subdomains [2]. In case of a three time levels scheme the linearized transport equation for variable ϕ at the n^{th} time step can be written as

$$A^n \phi^n + B \phi^{n-1} + C \phi^{n-2} = q^n \,, \tag{1}$$

where A^n denotes the coefficient matrix resulting from spatial discretization and linearization whereas B and C are determined by the time discretization scheme. In serial processing of successive time steps the solution of previous time steps are known and simply contribute to the source term q^n.

Within the time parallel approach, however, the variable values of k successive time steps are considered simultaneously as unknowns, which leads to an enlarged system of equations:

$$\begin{aligned}
A^n \phi^n + B \phi^{n-1} + C \phi^{n-2} &= q^n \\
A^{n-1} \phi^{n-1} + B \phi^{n-2} + C \phi^{n-3} &= q^{n-1} \\
&\vdots \\
A^{n-k} \phi^{n-k} + B \phi^{n-k-1} &= q^{n-k} - C \phi^{n-k-2} \\
A^{n-k-1} \phi^{n-k-1} &= q^{n-k-1} - B \phi^{n-k-2} - C \phi^{n-k-3}
\end{aligned} \tag{2}$$

In the last two equations, contributions from time steps with already fixed solutions has been brought to the right hand side.

The simplest domain decomposition in time is to solve for each time step concurrently, using available estimates of solutions of previous time steps (one outer iteration lagged). Thus, in contrast to space parallelism, domain decomposition in time only affects the assembley of source terms in outer iterations. The update of intermediate contributions to the source term requires large volume local communication of one-dimensional topology.

The efficiency of parallelized algorithms is assessed by analyzing the numerical, parallel and load-balancing efficiency, for details see [1]. In order to compare the numerical efficiencies for different parallelization strategies, the qubic lid-driven cavity flow with an

Mode	Processors space × time	Numerical efficiency (%)
serial	1 × 1	100
space parallel	4 × 1	97
	16 × 1	88
time parallel	1 × 4	100
	1 × 8	90
	1 × 12	72
combined	4 × 4	97

Table 1
Numerical efficiencies for different parallelization strategies. Note the high efficiency for 16 processors with combined space and time parallelism compared to the pure space parallel case.

oscillating lid ($Re_{max} = 10^4$) has been simulated on a 32^3 grid using up to 16 processors. In the serial run, approximately ten outer iterations per time step are necessary to reduce the sum of absolute residuals for all transport equations by six orders of magnitude. If the number of concurrent time levels exceeds the number of outer iterations per time step in a serial run, the numerical efficiency drops sharply, i.e. the number of outer iterations increases. This can be seen from the drop in the numerical efficiency between eight and twelve time parallel processors in Table 1. However, below that critical number of time parallel levels the numerical efficiency is high.

The benefit of time parallelism becomes clear by comparing the numerical efficiencies obtained using 16 processors: 88% for parallelism in space alone and 97% for the combined space and time parallel approach. Although not scalable, time parallelism can be efficiently combined with space parallelism for problem sizes too small to be efficiently decomposed in space using large number of processors. A more detailed description of numerical results (including multigrid application) is given in [3].

As long as the number of control volumes does not change in time the load between time parallel processors is automatically balanced, what can help to reduce the complexity involved in load balancing in space.

Because decomposition in space leads to fine-grain communication patterns, the set-up time to initialise communication (latency) appears to be the crucial hardware parameter with regard to parallel efficiency [1]. The coarse-grain communication of time parallelism, in contrast, requires high transfer bandwidths. However, the communication to update the time contribution to source terms can be overlapped with the assembley of the coefficient matrix. It will therefore depend on the hardware features whether one can benefit from time parallelism to obtain high total efficiencies. The possibilities to increase the parallel efficiency by overlapping computing and communication are analyzed in [4].

3. Parallel Preconditioned Conjugate Gradient Method

The relaxation of sparse linear systems is crucial to the whole application because it consumes a major part of the computing time and determines the numerical efficiency of the space parallel approach. Here we consider Conjugate Gradient (CG) methods preconditioned by incomplete lower upper (ILU) decomposition schemes, like Incomplete Cholesky (IC).

The basic iterative ILU scheme is parallelized by local decomposition of the global

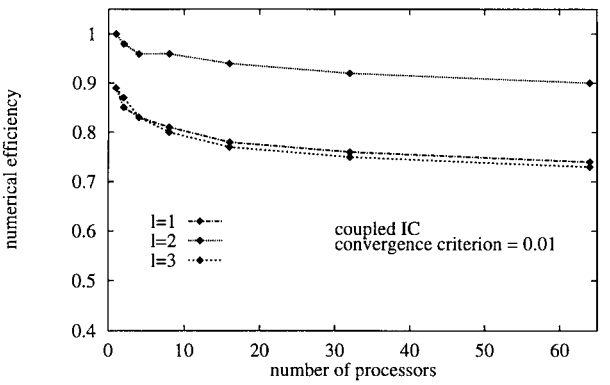


Figure 1. Numerical efficiency for IC preconditioned CG method. As convergence criterion the scaled sum of absolute residual has been used.

coefficient matrix A, according to the domain decomposition in space. A is thus split into a local part A_l and a coupling part A_c:

$$A = A_l + A_C\,, \tag{3}$$

where A_l has block diagonal structure for domainwise numbering of unknowns. Full concurrency is now forced by incomplete factorization of A_l instead of A, with the coupling part being explicitly calculated using the storage overlap between subdomains which is updated after each inner iteration.

Global communication due to global reductions of inner products constitutes a bottleneck of CG methods, especially on hardware platforms with long set-up times like workstation clusters. Therefore, restructured variants of the standard CG methods are used, which require less global communication events at the expense of additional inner products [5,3]. In addition, multiple basic iterations are combined with one CG acceleration. The resulting algorithms are known as l-step basic iterative preconditioned CG methods, where l denotes the number of basic iterations per CG sweep [6].

Figure 1 shows the numerical efficiency of an IC-preconditioned CG method for various values of l and numbers of (space) parallel processors. A Poisson equation with Neumann boundary conditions and low frequency harmonic source terms, which models the pressure-correction equation of the SIMPLE algorithm, has been relaxed on a non-uniform 64^3 grid.

Clearly, $l = 2$ is the optimum choice with regard to numerical efficiency. Note also that this variant seems to be the most robust with increasing number of processors, as its efficiency drops only by 10% over the given range of processors. The others drop by more than 15% and start from a lower level, since $l = 2$ is the optimum choice also for a single processor. Even for 512 processors, $l = 2$ still yields a numerical efficiency of 79%, whereas $l = 1$ drops down to 31%. The same trends have been observed for the preconditioned CGstab method for non-symmetric matrices [3].

Apart from its easy implementation, the described approach to parallize preconditioned CG methods therefore preserves the numerical efficiency while reducing global communication.

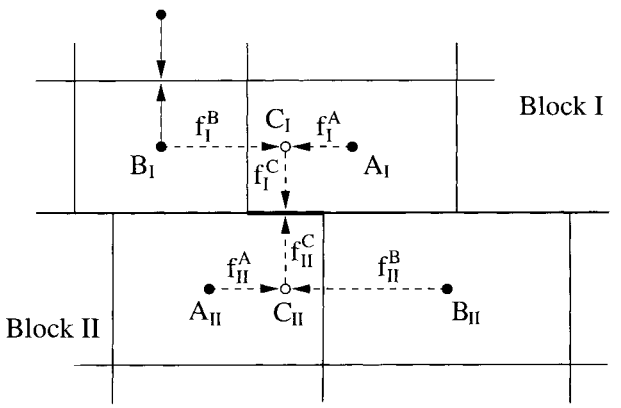


Figure 2. Illustration of discretization scheme at non-matching block interface.

4. Block-Adaptive Grids

Block-structured grids are a compromise between fully structured and fully unstructured grids. In that sense they combine efficiency based on regularity with flexibility. Apart from this, the block structure serves as a natural basis for domain decomposition in space.

In order to exploit the full capacity of the block-structured approach, one must allow for non-matching interfaces between grid blocks, which enables local grid refinement in regions where high variations of gradients are expected. This kind of grid will be referred to as block-adaptive grid.

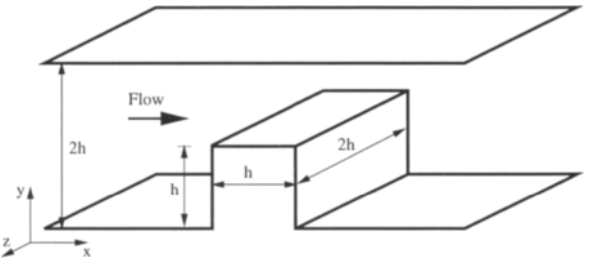
Figure 2 illustrates the discretization scheme for a two dimensional grid at a non-matching block interface. We define each interface segment common to two control volumes belonging to block 1 and 2 respectively as a block cell face. The fluxes through the block cell faces can be calculated in the same way as for any interior cell face by introducing auxiliary nodes C_I and C_{II} on either side. The value of any variable ϕ at these nodes is expressed through the neighbour nodes using linear interpolation:

$$\phi(C) = f^A \cdot \phi(A) + f^B \cdot \phi(B)\,, \tag{4}$$

where f^A and f^B are geometrical weights (with $f^A + f^B = 1$) calculated by the preprocessor. Thus, the discretization scheme remains second order accurate and fully conservative.

However, control volumes along interfaces now have more than four (in 2D or six in 3D) cell faces, which in an implicit solution method leads to enlarged computational molecules. The global coefficient matrix therefore becomes irregular. Choosing, however, non-matching interfaces to be subdomain boundaries of a domain decomposition preserves the regularity of iteration matrices as contributions from neighbour domains are treated explicitly. In conjunction with communication routines for arbitrary domain topologies, the above described strategy helps clustering grid points around obstacles without wasting them elsewhere. The following section will give an example (see Figure 4).

Figure 3. Physical configuration for the flow over a wall mounted rib in a channel. The streamwise length of the computational domain was $31\,h$. At walls the no-slip condition is imposed and periodic boundary conditions are applied in streamwise and spanwise directions with a fixed streamwise pressure drop.



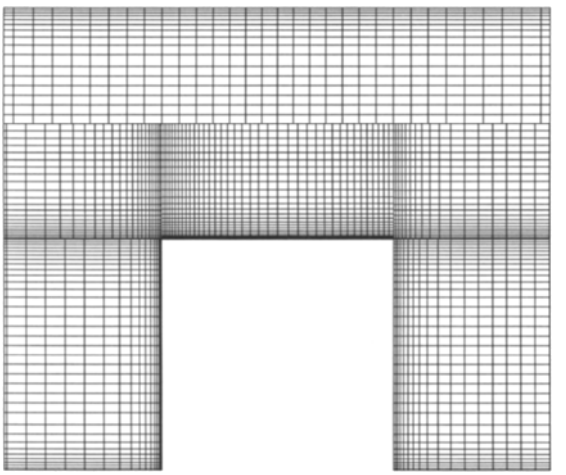


Figure 4. Section of the computational grid. The entire grid consists of $1.13 \cdot 10^6$ CV with 64 CV uniformly spaced in the spanwise direction. The centers of CV adjacent to horizontal channel walls are located at $\Delta y = 4.7 \cdot 10^{-3}$ and $\Delta y = 2.2 \cdot 10^{-3}$ from the top of the obstacle. On vertical walls the first grid points are at $\Delta x = 2.5 \cdot 10^{-3}$ from the left wall and $\Delta x = 3.8 \cdot 10^{-3}$ from the right wall.

5. Direct Numerical Simulation of Turbulent Channel Flow over a Square Rib

Figure 3 shows the physical configuration with a description of boundary conditions. Details of the computational grid are described in the context of Figure 4 . The simulation has been performed on the Cray T3D SC 192 at the Konrad-Zuse-Zentrum für Informationstechnik in Berlin with 32 processors parallel in space and one or two time parallel levels depending on the scheduling of the machine. The FORTRAN code achieves approximately 10 Mflops on a single processor and scales to $\sim$ 550 Mflops on 64 processors.

The developed flow has a Reynolds number of $Re = 3.7 \cdot 10^3$ based on the mean velocity U_m above the obstacle and the step height h. The simulation covers $\sim$ 87 characteristic time units $T_c = h/U_m$ with a 2D solution as initial field and $\sim$ 54 time steps per T_c. Results are averaged in the homogeneous direction and in time over the last 37 T_c. Yang

Figure 5. Instantaneous contours of streamwise velocity component u in a vertical cross-section. Velocities have been scaled with U_m.

and Ferziger [7] have simulated the same configuration with $Re = 3.21 \cdot 10^3$, which allows to some extent the comparison of the results.

A snapshot of the instantaneous contours of the streamwise velocity component u is given in Figure 5 . The flow develops several seperation and reattachment zones near the obstacle, which are illustrated in Figure 6 by the schematic contours of $U = 0$ (where U denotes the averaged streamwise velocity component). The corresponding seperation and reattachment lengths are also listed. Both the zone pattern and the principal reattachment lengths are consistent with results presented in [7].

Figure 7 shows profiles of the mean streamwise velocity U and its fluctuations u'^2 one step height behind the obstacle. Again, the results compare well with those from [7].

6. Summary

The solution method using block-adaptive cartesian grids presented here is suitable for simulation of flows in rectangular geometries. The parallelization strategy is simple and, through the combination of domain decomposition in both space and time and an

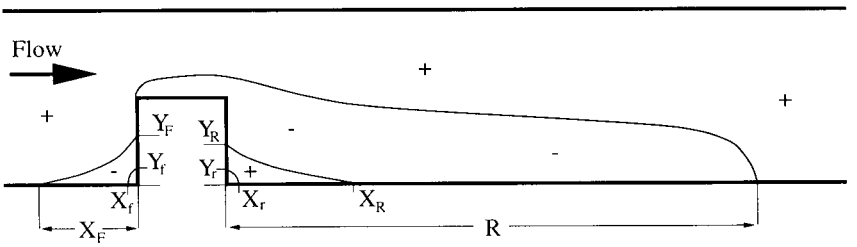


Figure 6. Seperation and reattachment zones (all lengths are given in step heights): $R = 6.4$, $X_R = 1.3$, $Y_R = 0.41$, $X_r = 4.6 \cdot 10^{-2}$, $Y_r = 6.5 \cdot 10^{-2}$, $X_F = 1.2$, $Y_F = 0.50$, $X_f = 4.4 \cdot 10^{-2}$, $Y_f = 7.5 \cdot 10^{-2}$.

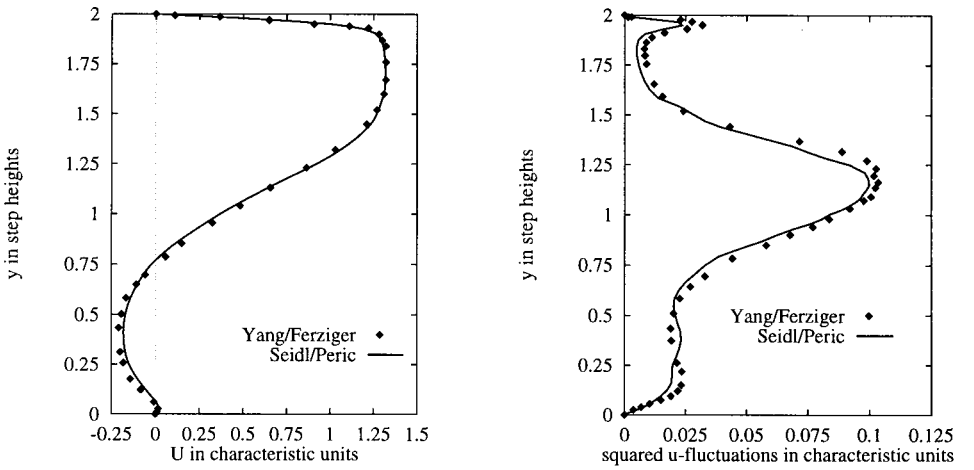


Figure 7. Mean streamwise velocity U and squared velocity fluctuations u'^2 profiles one step height behind the obstacle. Velocities have been scaled with U_m.

optimized linear equation solver also efficient. Results of a direct numerical simulation of flow over a wall mounted rib in a channel compare well with other simulations from literature.

REFERENCES

1. E. Schreck and M. Perić, Computation of Fluid Flow with a Parallel Multigrid Solver, *Int. Journal for Num. Methods in Fluids*, Vol. **16**, 303-327 (1993)
2. J. Burmeister and G. Horton, Time-Parallel Multigrid Solution of the Navier-Stokes Equations, *International Series of Numerical Mathematics*, Vol. **98** (1991)
3. V. Seidl and M. Perić, Internal Report, Institut für Schiffbau, Universität Hamburg (can be mailed on request)
4. E. Schreck and M. Perić, Analysis of Efficiency of Implicit CFD Methods on MIMD Computers, *this Volume*
5. G. Meurant, Domain Decomposition Methods for Solving Large Sparse Linear Systems, *NATO ASI Series F*, Vol. **77**, 185-206 (1991)
6. L. Adams, m-Step Preconditioned Conjugate Gradient Methods, *SIAM J. Sci. Stat. Comput.*, Vol. **6**, No. 2 (1985)
7. K. Yang and J. Ferziger, Large-Eddy Simulation of Turbulent Obstacle Flow Using a Dynamic Subgrid-Scale Model, *AIAA Journal*, Vol. **31**, No. 8 (1993)

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A Newton-GMRES Method for the Parallel Navier-Stokes Equations

J.Häuser[a], R.D. Williams[b], H.-G. Paap[c], M. Spel[d], J. Muylaert[d] and R. Winkelmann[a]

[a]Center of Logistics and Expert Systems, Salzgitter, Germany
[b]California Institute of Technology, Pasadena, California
[c]Genias GmbH., Regensburg, Germany
[d]ESTEC, Noordwijk, The Netherlands

1. INTRODUCTION

CFD is becoming increasingly sophisticated: grids define highly complex geometries, and flows are solved involving very different length and time scales. The solution of the Navier-Stokes equations has to be performed on parallel systems, both for reasons of overall computing power and cost effectiveness.

Complex geometries can either be gridded by completely unstructured grids or by structured multiblock grids. In the past, unstructured grid methods almost exclusively used tetrahedral elements. As has been pointed out in [1] and recently in[2] this approach has severe disadvantages with regard to program complexity, computing time, and solution accuracy as compared to hexahedral finite volume grids. Multiblock grids that are unstructured on the block level, but structured within a block provide the geometrical flexibility and retain the computational efficiency of finite difference methods.

In order to have the flow solution independent of the block topology, grids are slope continuous, and in the case of the N-S solutions an overlap of two points in each coordinate direction is provided, with a consequent memory overhead: if N is the number of internal points in each direction for a given block, this overhead is the factor $(N+4)^3/N^3$. The overhead is caused by geometrical complexity, i.e. to generate a block topology that aligns the flow with the grid as much as possible [3].

Since grid topology is determined by both the geometry and the flow physics, blocks are disparate in size, and hence load balancing is achieved by mapping a group of blocks to a single processor. The message passing algorithm has to be able to handle efficiently the communication between blocks that reside on the same processor, that is, there is only one copy operation involved. Message passing (PVM or MPI) is restricted to a handful of functions that are encapsulated, and thus full portability is achieved. In addition, the code is written in ANSI-C, which guarantees portability and provides significant advantages over Fortran (see for example [4]). A serial machine is treated as a 1 processor parallel machine without message passing. Therefore the code, *Parnss* [5], will run on any kind of architecture. Grids generated by *GridPro* [7] or *Grid**[1] (Plot3D format) can be directly used.

Available parallelism (the maximum number of processors that can be used for a given problem) is determined by the number of points in the grid, but a tool is avail-

able to split large blocks if necessary.

Regarding the solution algorithm for the Navier-Stokes equations, an explicit algorithm is easiest, but is not as efficient as relaxation schemes in calculating the steady state. Often relaxation schemes are used, but it should be remarked that even these methods may not converge for highly stretched grids with large aspect ratios (10^6), as needed in most viscous flows.

Thus we shall use implicit methods for these grids, with a linear solver chosen from the Krylov family. To make good use of these techniques, we need an effective and efficient preconditioner, and this paper is about the convergence properties of the Navier-Stokes code for different preconditioners used on parallel architectures.

2. Solving the N-S Equations

An implicit step of the discretized N-S equations can be cast in the form of a set of nonlinear equations for the flow variables, whose numerical solution is by a Newton scheme. The computational kernel is a linear solve with the matrix being the Jacobian of the equations. There are numerous schemes for such solves, such as Jacobi relaxation or lower-upper triangular split (Gauss-Seidel). As mentioned above, these schemes are slow to converge for a stiff system, caused by widely varying temporal and spatial scales.

The numerical solution proceeds in five major stages. In this work, stage 4 will be described in some detail.

1. *Topology:* Perform domain decomposition of the solution domain.
2. *Grid generation:* Create a high-quality grid within each domain. Spatial discretization reduces the N-S equations to a set of ODE's.
3. *Explicit Solution:* Advance explicitly in time by using a two step Runge-Kutta scheme.
4. *Implicit Solution:* Advance the solution implicitly in time with the backward Euler scheme, thus requiring solution of nonlinear equations, which can be solved by a Newton or quasi-Newton algorithm, which in turn requires solving sets of linear equations: we use preconditioned GMRES for these.
5. *Root Polishing:* For steady state solution, use a Newton iteration to derive the steady state, which is equivalent to an implicit step with infinite time step.

Investigations are underway to determine if it is possible to relax the accuracy with which the nonlinear equations in (4) are solved, yet still obtain robust and accurate convergence through stage (5).

2.1. Krylov Subspace Methods

In the following the CG method is described, because it is the basis for the *Generalized Minimal Residual (GMRES)* technique, as used in this paper, and for example, in [10]. However, the CG method will be presented mainly from a geometrical point of view to provide the motivation and insight in the workings of the method.

2.2. Conjugate Gradient

We have a system of linear equations, $\mathbf{Ax} = \mathbf{b}$, derived from an implicit step of the N-S equations, together with an initial solution vector $\mathbf{x}^0$. This initial vector may be obtained by an explicit step, or may be simply the flow field from the previous step.

We can write this linear system as the result of minimizing the quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}$$

where the gradient of f is **Ax** - **b**. In the CG method a succession of search directions $\mathbf{p}^m$ is employed -- how these directions are constructed is of no concern at the moment -- and a parameter α_m is computed such that $f(\mathbf{x}^m - \alpha_m\mathbf{p}^m)$ is minimized along the $\mathbf{p}^m$ direction. Upon setting $\mathbf{x}^{m+1}$ equal to $\mathbf{x}^m - \alpha_m\mathbf{p}^m$, the new search direction has to be found.

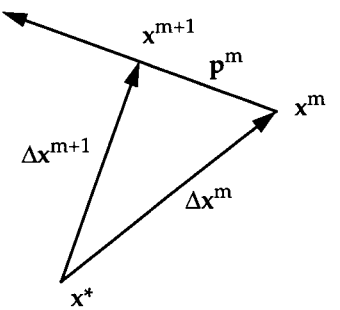


Figure 1. The CG method: one-dimensional minimization. Let $\mathbf{x}^*$ denote the exact (unknown) solution, $\mathbf{x}^m$ an approximate solution, and $\Delta\mathbf{x}^m = \mathbf{x}^m - \mathbf{x}^*$. Given any search direction $\mathbf{p}^m$, the minimal distance from the line to $\mathbf{x}^*$ is found by constructing $\Delta\mathbf{x}^{m+1}$ perpendicular to $\mathbf{p}^m$. Since the exact solution is (of course) unknown, we make the residual perpendicular to $\mathbf{p}^m$. Regardless how the new search direction is chosen, the norm of the residual is not increasing.

The construction of the search directions can be directly seen from Fig. 2.

In two dimensions, the contours f(**x**)=const form a set of concentric ellipses whose common center is the minimum of f(**x**). It can be shown that the residual vectors $\mathbf{r}^m$ form an orthogonal system and that the search vectors $\mathbf{p}^m$ are mutually **A**-orthogonal. The CG method has the major advantage that only short recurrences are needed, that is, the new vector $\mathbf{x}^m$ depends only on $\mathbf{x}^{m-1}$ and search direction $\mathbf{p}^m$. In other words, storage requirements are low.

The number of iterations of *CG* needed to achieve a prescribed accuracy is proportional to the square root of the *condition number* κ of the matrix, which is defined as the ratio of the highest to the lowest eigenvalue. Note that for second-order elliptic problems, κ increases by a factor of four when the grid-spacing is halved.

2.3. GMRES

Since the matrix obtained from the N-S equations is neither symmetric nor positive definite, the term $(\mathbf{p}^m, \mathbf{A}\mathbf{p}^m)$ is not guaranteed to be positive, and also the search vectors fail to be mutually orthogonal. It should be remembered that $\mathbf{p}^{m+1} = \mathbf{r}^m + \alpha_m\mathbf{p}^m$ and that the α_m are determined such that the second orthogonality condition holds.

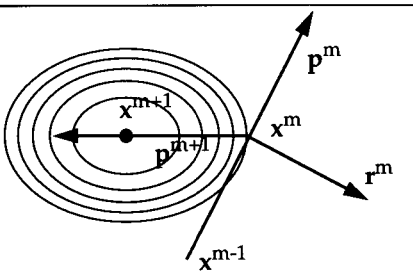


Figure 2. The CG method: Search directions. The next search direction $\mathbf{p}^{m+1}$ is **A**-orthogonal, or conjugate, to $\mathbf{p}^m$, and it is a linear combination of $\mathbf{r}^m$ and $\mathbf{p}^m$. This determines $\mathbf{p}^{m+1}$. In two dimensions, this second search direction goes through the midpoint of the ellipse, giving an exact solve at the second stage. Note that simply setting $\mathbf{p}^{m+1} = -\mathbf{r}^m$, the method of steepest descent, would not result in the most efficient search direction.

This is no longer possible for the N-S equations. However, this feature is mandatory to generate a basis of the solution space. Hence, this basis must be explicitly constructed. The extension of the CG method, termed *GMRES* (Generalized Minimized Residual), minimizes the norm of the residual in a subspace spanned by the set of vectors $\mathbf{r}^0$, $\mathbf{A}\mathbf{r}^0$, $\mathbf{A}^2\mathbf{r}^0$, ..., $\mathbf{A}^{m-1}\mathbf{r}^0$, where vector $\mathbf{r}^0$ is the initial residual, and the m-th approximation to the solution is chosen from this space. The above mentioned subspace, a *Krylov space*, is made orthogonal by the well known *Gram-Schmidt* procedure, known as an *Arnoldi* process when applied to a *Krylov* subspace.

When a new vector is added to the space (multiplying by **A**), it is projected onto all other basis vectors and made orthogonal with the others. Normalizing it and storing its norm in entry $h_{m,m-1}$, a matrix $\mathbf{H}_m$ is formed with nonzero entries on and above the main diagonal as well as in the subdiagonal. Inserting the ansatz for $\mathbf{x}^m$ into the residual equation, and after performing some modifications, a linear system of equations for the unknown coefficients γ_l^m involving matrix $\mathbf{H}_m$ is obtained. $\mathbf{H}_m$ is called an upper *Hessenberg* matrix. To annihilate the subdiagonal elements, a 2D rotation (*Givens rotation*) is performed for each column of $\mathbf{H}_m$ until $h_{m,m-1} = 0$. A Givens rotation is a simple 2x2 rotation matrix. An upper triangular matrix $\mathbf{R}_m$ remains, that can be solved by backsubstitution.

3. Preconditioners

In order to reduce the condition number of **A**, the system is premultiplied by a so called preconditioning matrix **P** that is an approximation to $\mathbf{A}^{-1}$, but is easy to compute. Instead of solving the sparse linear system **Ax=b**, the equivalent system **(PA)x=Pb** is solved. The choice of an effective (less iterations) and an efficient (less computer time) preconditioner is crucial to success of *GMRES*.

For the preconditioning to be *effective*, **P** should be a good approximation of $\mathbf{A}^{-1}$, so that the iterative methods will converge fast. For *efficiency*, the memory overhead and the additional cost per iteration should be small.

Any kind of iterative scheme can be used as a preconditioner, for instance, Jacobi or Gauss-Seidel relaxation, Successive Overrelaxation, Symmetric Successive Overrelaxation (SSOR), Red-Black or Line Gauss-Seidel Relaxation, Incomplete Lower-Upper Factorization (ILU). Thus the linear solver is a two-part process, with an inner loop of a simple iterative scheme, serving as the preconditioner for an outer loop of *GMRES*.

3.1. Preconditioners in *Parnss*

First we recall that the condition number, and hence the number of sweeps to convergence, of a grid increases dramatically as the grid is made finer, so we expect a good strategy is to use multiple grids at different resolutions, the coarser acting as a preconditioner for the finer.

In the following, however, we describe the various preconditioning matrices that have been constructed and implemented for use with the *Parnss* code on a fixed grid. In each case the preconditioner is made by repeating a simple iteration derived from a splitting of **A**:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{B}^{-1}\left[\,(\mathbf{B}-\mathbf{A})\,\mathbf{x}^{k}+\mathbf{b}\right]$$

The various forms of matrix **B** that have been implemented are based on splitting **A** into diagonal, lower-triangular and upper-triangular parts, **D**, **L** and **U**, respectively. They may be written:

- Diagonal: $\mathbf{B} = \mathbf{D}$
- Gauss-Seidel: $\mathbf{B} = \mathbf{D} - \mathbf{L}$
- Successive Overrelaxation: $\mathbf{B} = \mathbf{D}/\omega - \mathbf{L}$
- Symmetric Successive Overrelaxation: $\mathbf{B}_1 = \mathbf{D}/\omega - \mathbf{L}$ alternating with $\mathbf{B}_2 = \mathbf{D}/\omega - \mathbf{U}$

In contrast to the CG method, GMRES does not generate short recurrences, but needs the full set of basis vectors, thereby incurring a substantial memory overhead. Second, the computational cost increases linearly with the number of basis vectors. Since the code is written in ANSI-C, memory is allocated and freed dynamically. Moreover, only one block at a time is computed per processor, so that only this block has to be stored. In principal, the Krylov basis could be made as large as the available memory allows, dynamically adding new basis vectors. However, because of the computational overhead and the effects of rounding error in the orthogonalization, the algorithm should be restarted: we have chosen to do this after 20 iterations.

3.2. Experimental Results

We conclude this section with some measurements of the effectiveness of the various preconditioners. Figure 3 shows the fall of the norm of the residual for a sample calculation, against wall-clock time. The block-diagonal preconditioning is a variant of diagonal, where 5x5 diagonal blocks of the matrix are inverted, and red-black preconditioning is a Gauss-Seidel process where the grid points have been ordered in a particular way. It is clear that the SSOR is the most efficient.

4. Results for NASA-ESA Huygens Space Probe

Parnss was used to perform several testcase computations for the Huygens space probe. This space probe is part of a joint NASA-ESA project and will be launched in

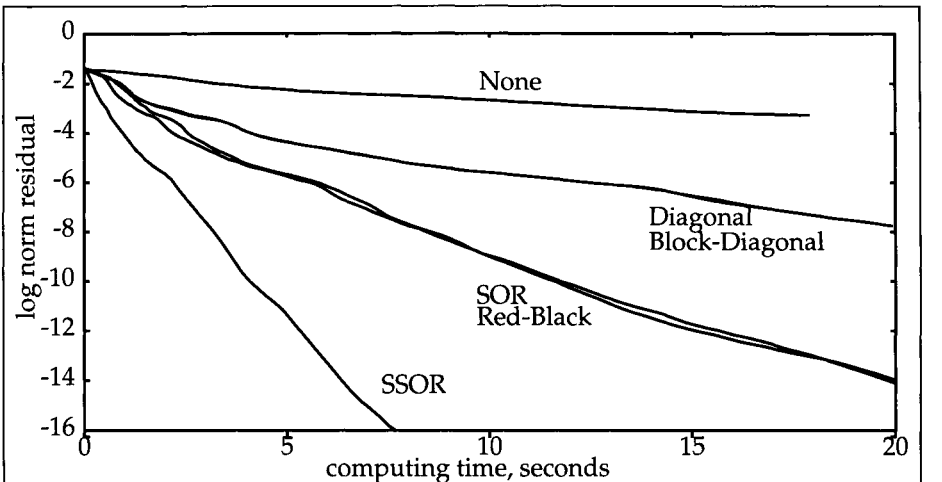


Figure 3: Effect of preconditioning. The example computed is for a 24 block 3D grid for the Huygens space probe. The plot shows the log magnitude of the residual versus computing time within one nonlinear iteration step.

1996. After a 6 year flight, Huygens will have reached Titan, the largest moon of Saturn, and upon completing the entry phase will descend by parachute (Mach 0.1) to measure the composition of Titan's atmosphere (mainly nitrogen). The concern is that sensors (lasers, mass spectrometer) located on the windward side of the probe may become inoperational if the local flow field is such that dust particles may be convected onto the optical surfaces of these sensors. So far, all computations have been for inviscid flow, but high cell aspect ratios were already used. In addition to the incompressible case, a Mach 3.1 computation has been performed. Computations were stopped when the residual had dropped to 5×10^{-6}. The computation for Ma 3.1 is not a realistic case, since the aerobraking shield was not modeled. For both low and high Mach numbers, the SSOR performed best.

5. Towards an Efficient Parallel Solution Strategy

In this paper we have briefly described a combined solution strategy for the N-S equations. Solutions for steady state as well for transient flow can be computed, covering the range from incompressible to hypersonic flows. The numerical solution technique is based on a Krylov subspace method. In particular, for very large and stiff problems the condition number will also be large, and preconditioning is essential. For these problems, conventional relaxation schemes converge very slowly or may not converge at all. A large number of preconditioners have been tested, and symmetric SOR has been found to work best. An excellent discussion of preconditioners for 2D test cases is given in [11].

From theoretical models [12] and computational experience [5] it can be concluded that load balancing for complex geometries and parallel efficiency up to several hundred processors do not pose a problem for the N-S simulations for the strategy outlined in this paper.

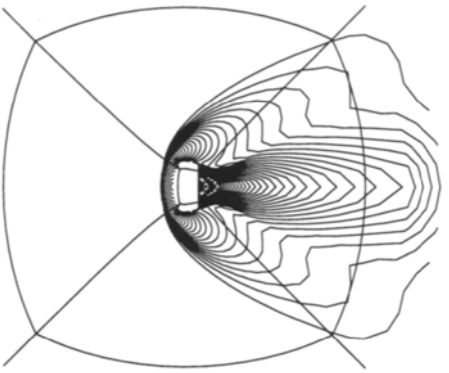


Figure 4: Mach distribution for a 6 block Huygens grid with AoA=0, Ma=3.1, and blocksize 19X61X19.

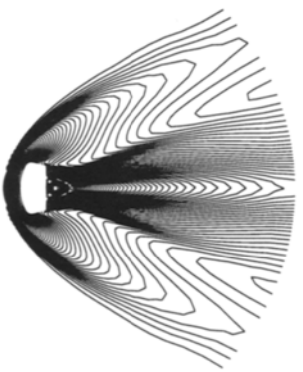


Figure 5: The same problem as in Fig. 4, but now the solution domain is split into 384 blocks of size 12x12x12, resulting in a total of 663,552 grid points. Although block size is small, communication overhead is not important because of the large amount of computation per grid point.

However, there is a need to investigate the convergence rate of these implicit solution schemes on serial and parallel architectures. The time-dependent N-S equations used to obtain the steady state are hyperbolic, that is, there is a finite propagation speed. A full numerical coupling of all linear equations is therefore unphysical, and actually reduces the convergence rate for steady state problems as large scale computations have shown. The 384 block Huygens space probe that was presented in Section 4 can also be modeled using a 6 block grid. For exactly the same problem, a much slower convergence to the steady state has been observed, exhibiting the same trend as in the 2D testcases [5]. Since the exact numbers have not been firmly established at

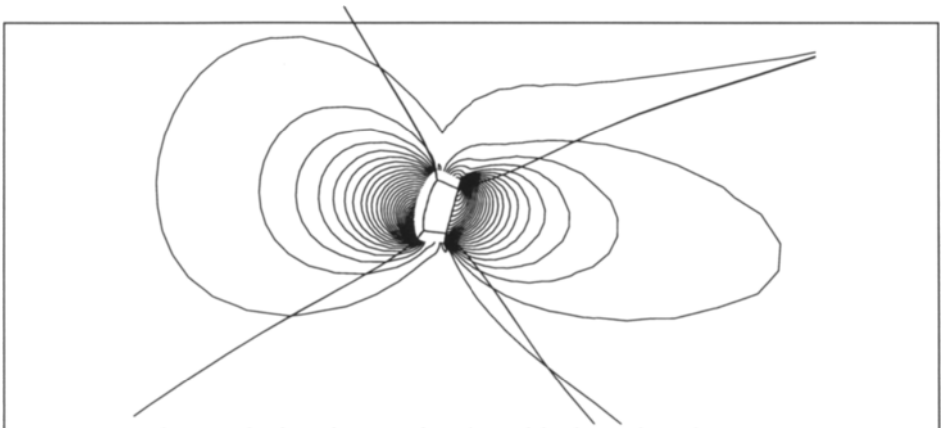


Figure 6: The Mach distribution for the 6-block grid with AoA=15, Ma=0.1 and blocksize 19x19x19.

the time of this writing, they will not be presented here.

6. REFERENCES

1. J. Häuser, J. Muylaert, H.-G. Paap, M. Spel, and P.R. Eiseman, Grid Generation for Spaceplanes, 3rd Space Course, University of Stuttgart, Germany, 1995, 66 pp.
2. D. J. Mavriplis and V. Venkatakrishnan, A Unified Multigrid Solver for the Navier-Stokes Equations on Mixed Element Meshes, AIAA-95-1666.
3. D. A. Kontinos, McRae, Rotated Upwind Algorithms for Solution of the Two- and Three-Dimensional Euler and Navier-Stokes Equations, AIAA 94-2291
4. J. Häuser, M. Spel. J. Muylaert and R. D. Williams, *Parnss:* An Efficient Parallel Navier-Stokes Solver for Complex Geometries, AIAA 94-2263.
5. J. Häuser, J. Muylaert, M. Spel, R. D. Williams and H.-G. Paap, Results for the Navier-Stokes Solver *Parnss* on Workstation Clusters and IBM SP1 Using PVM, in Computational Fluid Dynamics, Eds. S. Wagner et al., Wiley, pp. 432-442.
6. M. J. Bockelie and P.R. Eiseman, A Time Accurate Adaptive Grid Method for the Numerical Simulation of a Shock-Vortex Interaction, NASA-2998, 1990.
7. P. R. Eiseman, et al., *GridPro/az 3000*, Users's Guide and Reference Manual, 111 pp., Program Development Corporation of Scarsdale, NY.
8. D. Whitfield, Newton-Relaxation Schemes for Nonlinear Hyperbolic Systems, Mississippi State University Preprint MSSU-EIRS-ASE-90-3, 1990.
9. K. J. Vanden, Direct and Iterative Algorithms for the Three-Dimensional Euler-Equations, Dissertation Thesis, Mississippi State University, December 1992.
10. G. Golub, G., J.M. Ortega, Scientific Computing, Academic Press, 1993.
11. K. Ajmani, W. F. Ng, M. S. Liou, Preconditioned Conjugate-Gradient Methods for the Navier-Stokes Equations, J. of Comp. Phys., **110** (1994) 68-81.
12. J. Häuser, and R. D. Williams, Strategies for Parallelizing a Navier-Stokes Code on the Intel Touchstone Machines, Int. J. Num. Meth. Fluids, **15** (1992) 51-58.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A Parallel, Globally-Implicit Navier-Stokes Solver for Design

Zhen Zhao and Richard B. Pelz*
Mechanical and Aerospace Engineering,
Rutgers University, Piscataway, NJ 08855-0909, U.S.A.

Shape design puts severe new constraints on parallel flow solvers. With this in mind, we have developed a general, parallel, Euler and Navier-Stokes solver which uses a high-order upwind finite volume scheme, a global implicit iterative method coupled to a local implicit preconditioner, new flux limiters and a new remeshing scheme for changing body shapes. The code runs without change on single processor workstations and on multiprocessors such as the nCUBE and IBM SP2. All communication, scheduling, and boundary conditions are handled by an input logic file.

1. Introduction

The goal of our work is to produce a general multi-domain code that rapidly and repeatedly solves the steady or unsteady Euler or Navier-Stokes equations for changing configurations. It is to be used for design and optimization of shapes as part of a greater effort in multidisciplinary design and optimization. [2] The problem of finding the flow around numerous body shapes during one design cycle requires very large computational resources, i.e. parallel computers. While some parallelization may be found in the optimization method, it is clear that the flow solver must be executable in parallel.

An implicit method was chosen in order to effectively solve for viscous flows on fine meshes. Iterative schemes similar to the Conjugate-Gradient method can effectively solve the linear non-symmetric operator coming from an implicit treatment of the linearized Navier-Stokes equations. [3] We employ one of these, the Bi-CGSTAB [10] iterative method. To ensure rapid convergence, it is not only implicit within a block or subdomain, but is globally implicit, allowing the multiprocessor and single processor convergence rates to be the same. A local preconditioner based on an approximate factorization scheme [4] is used to speed convergence of the linear problem. The flow code is described in Section 2.

Because topologically different configurations for different design problems will be examined, the code must handle general configurations and run without change on a single processor as well as on a host of multiprocessors. We have isolated the logic that handles an arbitrary geometry, decomposition, boundary condition, communication and computation scheduling, removed it from the code and moved it to an instruction logic file. By simply interchanging the logic files, a different problem can be solved. The logic file and parallelization are discussed in Section 3.

*This work is supported by a grant from NASA Langley Research Center, NAG-1-1559

To prevent oscillation of the second or third order accurate flow solutions around shocks, a modification of a standard limiter [1] is developed. The modification isolates the limiter to shock regions only. To prevent limit cycles from occurring, the limiter is smoothed. One way to do this is to integrate implicitly the time-dependent heat equation with the dependent variable being the limiter switch. When executing this on multiprocessors, care should be taken to ensure that the smoothing is not dependent on the subdomain size. A globally implicit smoother for the limiter is developed and presented in Section 4.

In the design procedure, the flow field for one configuration is solved, the shape is changed, and the flow field is again required. Obviously, using the old flow field as the initial condition will speed convergence of the second solve if the shape change is small. What is less obvious is how to modify the grid to reflect the shape change while keeping a smooth variation in mesh aspect ratio and other qualities of the original grid. The grid could be regenerated for each shape change, but that would be wasteful since the shape change is, in general, local and a global adjustment is unnecessary. To avoid serial bottlenecks, the grid modification should also be automated and done in parallel. We present such a parallel method to realign the grid with the new body shape based on elasticity theory in Section 5.

2. Description of Solver

We solve the Navier-Stokes and Euler equations for the two-dimensional compressible calorically perfect fluid flow. The equations are in conservation form for a generalized curvilinear coordinate system, but due to space limitations we omit their presentation. We use the Crank-Nicolson scheme in time with the linearization of the flux Jacobian matrices about time level n to give the equation

$$A^n \Delta Q^n = R^n \tag{1}$$

where ΔQ^n is the change in the state vector from n to $n+1$, A = [I / Δ t + 1/2 (inviscid and viscous flux Jacobians)] and R^n is the residual.

For space differencing, a conservative upwind finite volume scheme ($\leq$ 3rd order) based on the flux-vector splitting scheme of Thomas and Walters [9] along with the Roe-split [7] version of the convective fluxes are employed. Central differences are used to handle the viscous terms.

Such a discretization leads to a sparse, block, banded, nonsymmetric linear system of rank $nx \cdot ny$. A global, implicit, iterative scheme and a local implicit preconditioner are used to solve the system.

There are many CG-like iterative schemes to solve nonsymmetric linear systems. A comparison of many of these methods can be found in Barrett et al. [3] One that maintains the three-term recurrence relation, which minimizes the memory requirements, and relaxes the residual-minimization property is Bi-CGSTAB. [10] It requires 2 matrix vector multiplications and 4 inner products per iteration.

We use a local, implicit, approximately factored scheme [4] as a preconditioner for Bi-CGSTAB. By local, we mean that the preconditioner is applied to each subdomain separately, assuming that variables located outside the subdomain do not change. This allows the preconditioner to be executed without communication. While there are many

preconditioners (see for instance [5]), we choose this one because of the availability of the AF code and the ease of portability.

3. Parallelization and Logic File

The parallelization is based on the decomposition of the computational domain into non-overlapping subdomains. Each processor is assigned to one or more subdomains. Assuming that the computational domain is divided equally into N subdomains, the number of processors will be between 1 and N. In order for the residual calculation to proceed in each subdomain independently, data the width of the stencil must be provided on the edges of each subdomain. This data comes from boundary conditions or from data transferred from neighboring subdomains.

The program can automatically switch between workstations, nCUBE-2 and IBM SP2 multiprocessors. No change in the solver is required when changing configurations, domain connectivity, number of subdomains, number of processors, etc. An interblock logic file provides all the necessary information for mapping, scheduling, boundary conditions and communications. It contains :

- subdomain and processor number:
  Each subdomain has a unique index number and a corresponding processor number. The mappings are found through a stochastic optimization procedure [2] to minimize the distance of the communicated processors and reduce the communication time.

- type of communication or boundary condition:
  write to buffer & send, receive & read from buffer, body, wake, inflow, outflow and corner boundary conditions,

- grid points involved in boundary conditions or communication,

- local subdomain number and processor number to which data are sent,

- reversed n-coordinate, m-coordinate, axes for grids that are not co-aligned.

This file is used to set the scheduling of boundary conditions and communication. Each grid decomposition and topology change requires a different logic file. This results in the flow code being frozen while new geometries can be run by constructing a new logic file.

There are three basic linear algebra operations in Bi-CGSTAB: matrix-vector products, inner products and scalar-vector products. In the parallel environment, computation of the inner products requires inter-processor communication since the local inner products have to be accumulated among all the processors. This can be done using $log_2 P$ communication of length one. The scalar-vector products can be performed without any communication. For the matrix-vector products, a five-point stencil (i.e. 1-layer halo communication) is used to assemble the coefficient matrix A (assuming first order differencing). The inter-processor communication of the components of the multiplying vector between neighbor processors is required, and the schedule is made by the logic file.

The implicit scheme is very robust, remaining stable for any CFL number and any configuration. The local preconditioner significantly reduces the number of iterations per

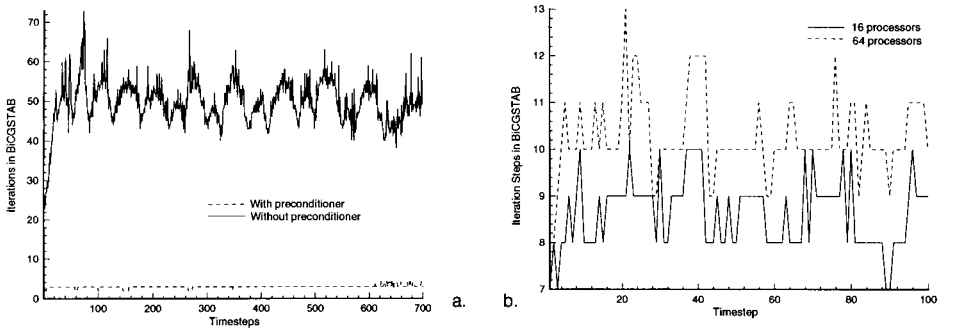


Figure 1. The number of iterations per timestep needed to reduce the linear residual by 2 orders versus the number of timesteps to reduce the nonlinear residual 8 orders. (64x64 mesh, subcritical flow, single element). Figure a: A comparison between the unpreconditioned and preconditioned Bi-CGSTAB algorithms for CFL=3. Figure b: A comparison between 16 and 64 processors for CFL=10.

timestep (see a. of figure 1). The expense of the preconditioner is about one residual time unit (RU) per iteration. The minimum CPU time to convergence (8 orders) occurs at about CFL=20 (see figure 2). It is not clear whether the spitting or the linearization error is limiting this CFL number to 20. Other preconditioners are currently being tested to answer this question.

Incomplete convergence at each timestep is also studied. The nonlinear convergence rate remains unaffected if the linear residual is reduced by two orders of magnitude or more. Since the preconditioner is local, its approximation to the original operator gets worse with decreasing granularity. We have, however, found this effect to be mild. The average number of iterations increases from 8 to 10 per timestep when the number of processors is increased from 16 to 64 for typical problems (see b. of figure 1).

4. Parallel Flux Limiter

As was discussed in the introduction, a limiter is used to prevent spurious oscillations around the shock when using a numerical method with accuracy greater than first order. We introduce a field variable ϕ which when equal to one allows a high order upwind differencing to be used, while when equal to zero, reduces the accuracy to first order to obey the Godunov theorem of monotone schemes.

Applying the van Albada limiter [1] on the pressure, the condition used to set the field variable to zero is

$$\frac{2\Delta \ln p_{i+1/2} \cdot \Delta \ln p_{i-1/2}}{(\Delta \ln p_{i+1/2})^2 + (\Delta \ln p_{i-1/2})^2} < \epsilon \tag{2}$$

where $\epsilon = .8$. While ϕ throughout the field is set to one, at the point (i, j) at which the

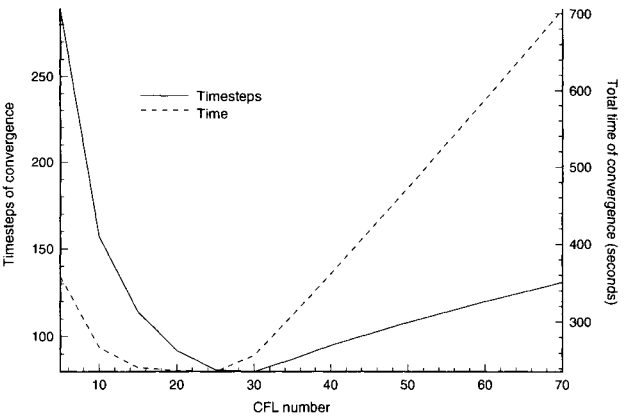


Figure 2. Timesteps and total time to convergence (8 orders) versus CFL number for a subcritical single element (64x64 mesh)

condition is true, $\phi(i,j)$ as well as $\phi(i\pm 1,j)$ and $\phi(i,j\pm 1)$ are set to zero. This operation is done independently on each processor

The spatial extent of the limiter should be small to prevent too much of the space being of low order. However, if the limiter is not wide enough or smooth enough ("enough" being problem dependent) the shock and limiter region may enter into a limit cycle preventing convergence. With this standard form of the limiter, gradients at the leading and trailing edges of a subcritical undisturbed flow were large enough to activate the limiter thus reducing those regions to first order. In some cases, the subsonic to supersonic line also became first order.

To alleviate the problem of the limiter being activated in nonshock regions, we developed the following modifications to the limiter.

$$\max(a_{i-1}, a_i, a_{i+1}) > 1 \; and \; \Delta \ln p_i \cdot \vec{v} > 0 \tag{3}$$

where a is the speed of sound. The first constraint ensures that only supercritical regions are activated. The second one allows only those regions with pressure drop in the same direction as the flow velocity to be first order.

To prevent the limit cycles from occurring, we smooth ϕ by taking one backward Euler step of the heat equation for the limiter [6]

$$\phi^{n+1} - \phi^n = \eta \Delta \phi^{n+1} \tag{4}$$

where the Laplace operator is in the computational plane and $\eta = 0.1$. This smooths the limiter in space. Since a 5 point stencil of ϕ is used to form the nonlinear residual, ϕ on the body must also be set. Homogeneous Neumann conditions on the body are used for the smoothing. To solve this linear system such that the subdomain boundaries do not

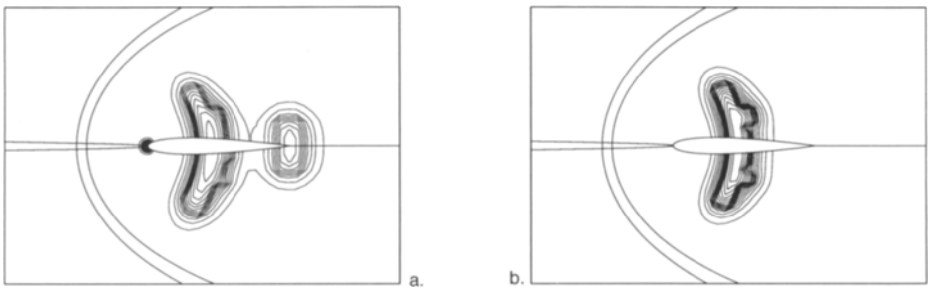


Figure 3. Contour plots of ϕ for transonic flow. Figure a. shows the smoothed unmodified limiter; the leading and trailing edge regions are first order. Figure b. shows the smoothed modified limiter.

affect the smoothness of ϕ, a globally iterative scheme is used. The operator is symmetric, so CG could have been used; however, the Bi-CGSTAB algorithm is used instead. The communication pattern is identical to that of the flow solver so the same logic file and schedule is used. Several iterations brought this linear residual down a few orders of magnitude. Convergence difficulties were not found.

In figure 3 we show contour plots of the flux limiter ϕ for a transonic single-element case. The left figure (a.) shows the smoothed unmodified limiter. Regions around the leading and trailing edges are first order. The right figure (b.) shows the smoothed modified limiter. Only the region (width 3-4 points) around the shock is first order. No problems were encountered in convergence to machine zero with the modified, smoothed limiter.

5. Parallel Grid Adjustment

In our projects of shape design, we use Simulated Annealing as an optimization method [2]. Here the set of design parameters which define the body shape are set, the flow and objective function are found. The SA algorithm selects a new set of parameters randomly (but within a neighborhood). The flow is solved, the objective function is found and compared to the previous one. The new body is accepted if the objective function is lower (for minimization problems), or if a hill-climbing criterion is met. These steps are then repeated as long as desired, perhaps hundreds of times.

The problem is to update the grid to go smoothly around the new body and to do it in parallel without much overhead. We first assume that the grid can be modeled as an elastic solid which is in equilibrium with the original body shape. The body shape change acts as a deformation of the boundary of the elastic solid. The subsequent change in the grid can be thought of as the relaxation of the elastic solid to a new equilibrium state with the new body shape. The equilibrium equation for the deformation $\vec{u}$ is

$$\nu \Delta \vec{u} + (\nu + \lambda) \nabla (\nabla \cdot \vec{u}) = 0 \tag{5}$$

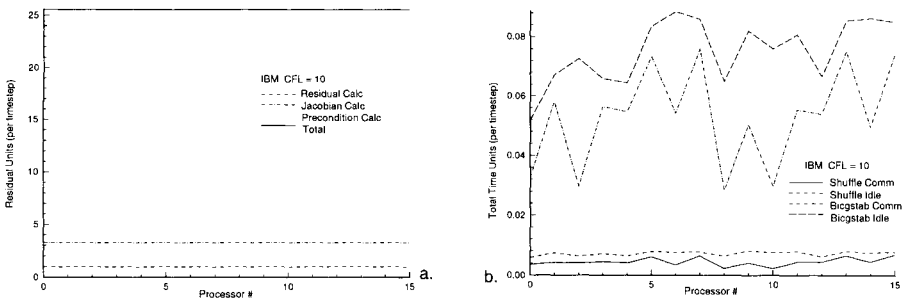


Figure 4. Times reported for each processor on the SP2, CFL = 10, P=16, local mesh 16x16. Figure a. shows incremental calculation times, and figure b. shows communications times.

Assuming only solenoidal deformations simplifies this to Laplace's equation with Dirichlet conditions on the boundary. The displacement is $\vec{u} = \vec{x}_{new} - \vec{x}_{old}$ with the displacement initially nonzero only at the boundary. The system can also be easily solved implicitly in parallel using the iterative solver of the code. The same logic file is used.

6. Results and Concluding Remarks

In figure 4 we show a breakdown of computation (a.) and communication (b.) times for a 16 processor run on the SP2 at NASA Langley, CFL=10. The results are incremental, i.e., the vertical height between lines reflects the time of the specific operation referenced by the upper line. The times are not averaged; each processor reports its own timings, which gives an indication of the load balancing. Note that the bulk of the CPU time occurs in the iterative solver; however, total CPU time to convergence is still less than with an explicit scheme. In the right plot of figure 4, the times are normalized by the total time per iteration. We see that total communication times for this relatively fine grain problem are less than 10%. The communication time in the iterative solver makes up the bulk of the total communication time.

We have described a general parallel Navier-Stokes solver with a global iterative / local preconditioned implicit scheme that runs on single and multiple processors. When changing geometries or machines, changes in a logic file, read at execution is all that is required. Flux limiter smoothing and grid adjustment also make use of the implicit scheme. An analogy of the mesh deformation and return to equilibrium of an elastic solid provides a parallel method for mesh regridding needed because of shape changes in a design and optimization loop.

Finally, we show the pressure contours of a converged solution for a two-element airfoil. The grid was severely skewed with mesh aspect ratios ranging 8 orders. The nCUBE-2 was used.

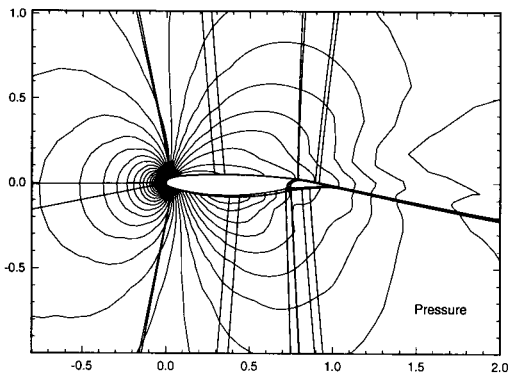


Figure 5. Pressure Contours around an airfoil with flap. M=.5, Re=1000, 32 subdomains and processors, nCUBE2, 16x16 local grids

REFERENCES

1. G.D. van Albada, B. van Leer and W.W. Roberts, Jr. "A comparative study of computational methods in cosmic gas dynamics." *Astronom. and Astropyhs.* 108:76-84, 1982.
2. S. Aly, F. Marconi, M. Ogot, R. Pelz and M. Siclari "Stochastic optimization applied to CFD shape design." *12th AIAA Computational Fluid Dynamics* 11-20, June, 1995.
3. R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. van de Vorst *Templates for the solution of linear systems: building blocks for iterative methods.* SIAM, Philadelphia, 1994.
4. R. M. Beam and R. F. Warming "An implicit factored scheme for the compressible Navier-Stokes equations." *AIAA Journal,* 16(4):393-402, 1978.
5. M. D. Kremenetsky, J. L. Richardson and H.D. Simon "Parallel Preconditioning for CFD Problems on the CM-5." *Parallel CFD* Ecer, Haeuser, Leca and Periaux, eds. North Holland, Amsterdam 1995.
6. F. Marconi private communication.
7. P. L. Roe "Discrete models for the numerical analysis of time-dependent multidimensional gas dynamics." *Journal of Computational Physics,* 63:458-476, 1986.
8. Y. Saad and M. Schultz "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems." *SIAM Journal on Scientific and Statistical Computing,* 7:856-869, 1986.
9. J.L. Thomas and R.W. Walters "Upwind Relaxation Algorithms for the Navier-Stokes Equations," *AIAA J.* 25(4):527-534, 1987.
10. H. A. Van der Vorst "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems." *SIAM Journal on Scientific and Statistical Computing,* 13(2):631-644, 1992.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel solution of viscous incompressible flow on multi–block structured grids using MPI

Ramesh Pankajakshan[a] and W. Roger Briley[b]

[a] Computational Engineering, [b] Mechanical Engineering
NSF Engineering Research Center for Computational Field Simulation, Mississippi State University, P.O. Box 6176, Mississippi State, MS 39762

1. ABSTRACT

A parallel code based on an existing unsteady three–dimensional incompressible viscous flow solver has been developed for simulation of flows past complex configurations using multi–block structured grids and using MPI for message passing. The linearized implicit solution algorithm used is modified for execution in parallel using a block–decoupled subiterative strategy, and a heuristic performance estimate is developed to guide the parallel problem definition. This performance estimate couples the parallel domain decomposition and grid generation with the software and hardware characteristics of the targeted computer system. The performance estimates obtained here for the IBM SP–2 generally agree with actual run times and seem useful for an *a priori* problem definition. Extensive evaluations of both algorithmic and parallel performance are given for a test problem consisting of axisymmetric flow past an unappended submarine hull with a Reynolds number of 12 million. Other results are given for three–dimensional appended submarine cases having 0.6M and 3.3M grid points. The decoupled subiteration algorithm used here allows the convergence rate of the sequential algorithm to be recovered at reasonable cost by using a sufficient number of subiterations, and allows decompositions having multiple subdivisions across boundary layers. The parallel algorithm is also insensitive to various orderings of the sweep directions during subiteration. Finally, the present approach has given reasonable parallel performance for large scale steady flow simulations wherein the grid and decomposition were appropriately sized for the targeted computer architecture.

2. INTRODUCTION

A primary objective of the present study is to develop an efficient parallel flow solver for large scale viscous incompressible flow simulations. The present study considers two– and three–dimensional steady flows, but the approach is also suitable for unsteady flows. A second objective is to implement and test this parallel flow solver for three–dimensional flow past submarine geometries. In view of this focus on issues arising in complex flow problems, the present parallel implementation is based on an evolving sequential code being developed and applied to a number of complex submarine flows, in the Computational Fluid Dynamics Lab at the ERC [1–4]. The

present study considers steady flow past submarine configurations (see Figure 1) including hull, sail with and without sail planes, and stern appendages. A future objective is to develop this code to predict trajectories of fully–configured maneuvering submarines including propulsors, based on hydrodynamic predictions of unsteady forces and moments.

The present approach exploits coarse–grained parallelism and message passing. The parallel implementation uses the Message Passing Interface (MPI) [5] for message passing. MPI was chosen for this study because of its portability, features which simplify the development of applications code, and its evolving status as an informal open standard with widespread support.

3. SOLUTION METHODOLOGY

3.1. Sequential Algorithm

In the sequential approach used for complex simulations (Taylor & Whitfield, [1–4]), a multi–zone grid with blocks of arbitrary size is first generated to fit the geometry and to provide adequate resolution of the flow structures of interest. The flow solver then uses an artificial compressibility formulation to solve the three–dimensional unsteady incompressible time–averaged Navier–Stokes equations for the flow within each of the grid blocks in a predetermined sequential order. The flow solver includes linearized subiterations and optional Newton iterations (for unsteady flows) at each time step. Typically for steady flows, the subiterations are completed *before continuing* to the next grid block in the sequence.

The time linearized approximations are solved by a subiteration at each time step. The subiteration procedure is a symmetric Gauss–Seidel relaxation (SGS) procedure [4], which can also be formulated as a lower–upper/approximate factorization (LU/AF) algorithm [6]. For steady flows, a local time step (or CFL number) is used to accelerate convergence to the steady solution. Other key elements of the solution methodology include high accuracy flux–based upwind finite–volume approximations in transformed coordinates. The spatial approximation combines Roe's approximate Riemann solver [7] and van Leer's MUSCL scheme [8,9]. The linearized flux Jacobians required by the implicit algorithm are computed by a numerical evaluation, as suggested in [2,4], and then stored and updated infrequently.

3.2. Parallel Algorithm

The SGS subiterations in the sequential algorithm are associated with good stability and convergence properties but are not easily parallelized. The approach followed here is to implement this algorithm on spatially decomposed grid blocks assigned to separate processes in a context such that the subiterations are effectively decoupled, so that each block can be done in parallel. This decoupling tends to give good parallel performance, but the decoupling of subiterations in blocks can degrade the algorithmic performance of this otherwise implicit algorithm.

One objective in studying this parallel algorithm is to identify a procedure that keeps the total number of arithmetic operations required for a converged solution as close as possible to that for the sequential algorithm. It will be demonstrated that the convergence rate of the sequential algorithm can be recovered at reasonable cost by using a sufficient number of (inexpensive) subiterations.

The parallel implementation employs an overlapping spatial domain decomposition of the grid into blocks which are assigned to separate processes. During each SGS

or LU/AF subiteration, the solution increments are exchanged for two rows (or surfaces) of points adjacent to each block interface. The decoupling is accomplished by starting each sweep of the subiteration within each block using boundary conditions from the previous subiteration sweep instead of waiting for values from the current subiteration. This algorithm uses Gauss–Seidel relaxation sweeps within each process but is effectively explicit across block boundaries, allowing for parallel solution for all blocks. The solution increments are updated by message passing following completion of the *forward* sweep of the subiteration and then again following the *backward* sweep. In addition, an algebraic eddy viscosity turbulence model is used (Baldwin & Lomax [10]) which requires global line searches that traverse block boundaries. The turbulence model is updated before each time step. Further discussion of the MPI implementation are given as an example application in [11].

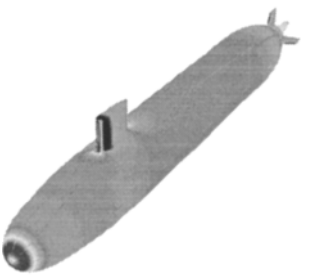


Figure 1. Pressure contours for submarine with sail and stern appendages

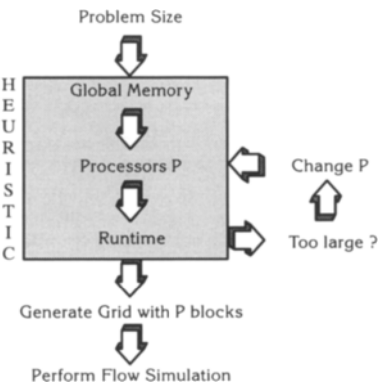


Figure 2. Approach for flow simulation on parallel machines

3.3. Problem Definition and Parallel Efficiency

Another objective of this study is to identify a spatial domain decomposition that leads to good parallel efficiency on the available computing platform. This is done by choosing a minimum number of processors having sufficient global memory for the case to be run, and arranging for a grid block structure with small surface to volume ratio to reduce communication, and with equal block sizes for load balance on homogeneous processors.

This procedure is illustrated in Figure 2. Once the CFD problem to be solved is defined, the problem size determines the global memory required, and this memory determines the minimum number of processors required on the available computing platform. The runtime is then estimated using a heuristic performance estimate (Section 4) based on both solution algorithm and architectural parameters. Communication/computation ratios are reduced by keeping the granularity as coarse as available memory allows (to give a large volume of data for the CPU), and by choosing a decomposition with small surface to volume ratio to reduce the size of messages. If the estimated runtime is too large, it can be reduced by changing the number of processors P with guidance from the performance estimate. The final step is to generate a grid having P blocks.

In effect, the parallel decomposition and load balancing issues are introduced as constraints in the grid generation process. These constraints are easily dealt with by experienced grid generators with good grid generation programs, and at some

point, the constrained grid generation may be automated. If an existing grid must be used, it can often be reblocked according to these guidelines. This constrained grid generation process yields a multi–block grid with blocks sized appropriately for the flow solver and computer configuration. A uniform grid block size provides static load balancing. The approach is well suited for problems that can be addressed by multi–block dynamic grids generated at the start of the flow calculation. The use of adaptive grids which require reblocking during the flow calculation has not been considered in this study.

4. PARALLEL PERFORMANCE ESTIMATE

A heuristic means of estimating the parallel performance, given key parameters for the solution algorithm and parallel computing platform, is outlined below.

4.1. Storage and CPU Requirements

Assuming an $(n_1 \times n_2 \times n_3)$ grid with N finite volume cells, the storage required for large arrays is approximately $248\,N$ 64–bit words. Thus, a 50,000 point grid needs 109Mb.

The floating point operations required by the solution algorithm during one step include three basic components of the calculation: (a) Residual evaluation, (b) LU subiteration sweeps, and (c) Numerical flux Jacobian linearizations. These were determined by a sequence of calibration runs on a Cray YMP and are summarized here in *MFLOP* (million floating point operations) as (a) Residual: $1230 \times 10^{-6}\ N\ \ MFLOP$, (b) Subiteration: $330.5 \times 10^{-6}\ N\ \ MFLOP$, (c) Jacobian: $5284.3 \times 10^{-6}\ N\ \ MFLOP$. If I_S denotes the number of LU subiterations, if the flux Jacobian linearizations are updated every I_J time steps, and if R_{CPU} is the effective processor speed in Mflops, then the total CPU time in seconds for an average step is given by

$$CPU\ (seconds) = \left[Residual + I_S \times Subiteration + \frac{Jacobian}{I_J} \right] \ / \ R_{CPU} \qquad (4.1)$$

4.2. Communication Requirements

Three basic components of the interprocessor communication requirement are considered: (a) exchange of data adjacent to block interfaces during subiteration, (b) global operations required by the algebraic eddy viscosity turbulence model, and (c) loading and unloading of buffer arrays used in message passing (a slightly different MPI implementation with user–defined datatypes would avoid this explicit buffer loading, but has not yet been implemented).

The communications estimate is expressed in terms of the following parameters: Number of Processors, P ; Number of Block Interfaces Exchanging Messages, N_m ; Message Length for a Single Data Surface (Mb), L_m ; MPI Software Latency (s), σ ; MPI Software Bandwidth (Mb/s), β . During each of the I_S subiterations, the solution time increment ΔQ must be exchanged twice (once for the forward sweep and once for the backward sweep) for each of the N_m block interfaces on each process. The overlapping domain decomposition is such that data is duplicated and exchanged for two surfaces of points adjacent to each block interface. In addition, the block decoupled solution algorithm is synchronized such that all messages are sent at approximately the same time. Consequently, an estimate for bisection width is needed to account for self contention in message passing. A factor of $\sqrt{P}$ is a suitable estimate and is exact for a two–dimensional mesh. The estimated total time required for message passing is given by

$$t_{comm} = 2\, I_S\, N_m \left[\sigma + \frac{2\, L_m}{\beta}\, \sqrt{P}\right] + 2 \left[q\, \sigma + \frac{2\, L_m}{\beta}\, \sqrt{P}\right] + 2\, I_S\, N_m \left[\frac{2\, L_m}{9.71}\right]$$

Point–to–point Global Buffering

where q is such that $P > 2^{q-1}$.

For the architectural parameters, the effective processor speed R_{CPU} was calibrated for an IBM SP–2 processor as approximately 49 Mflops, the MPI software latency and bandwidth were obtained from [12] for an IBM SP–2 as $\sigma = 62\ \mu s$ and $\beta = 34\ Mb/s$ (the asymptotic rate for large messages).

5. RESULTS

A two–dimensional flow case was used for extensive evaluation of both parallel and algorithmic performance. First, a study was done to determine algorithmic parameters that minimize the global operation count for a converged solution using the sequential algorithm. Next, results were obtained to identify guidelines for selecting the number of subiterations for the decoupled parallel algorithm to maintain the convergence rate of the sequential algorithm. Finally, the performance estimate is used to evaluate the problem definition and parallel efficiency for this same small test case and for much larger three–dimensional cases for appended submarine configurations.

5.1. Early Development

The portability of the MPI implementation was useful in that the initial development was done on a network of SUN workstations for grids of order 14,000 points. A much larger multi–block case (0.6M points) for turbulent flow past a submarine hull with sail and stern appendages was run on a twelve–processor SGI Challenge, and on both IBM SP–1 and SP–2 systems. For the 12 block, 0.6M grid case, 500 iterations required 6 hours on the twelve–processor SGI Challenge, with 92 percent of the runtime devoted to floating point calculations. The 12 node runs on the Challenge and the SP1 gave sustained performances of 84 MFLOPS and 120 MFLOPS respectively. Apart from using the highest compiler optimization level available, no effort has yet been made to tune the code.

5.2. Algorithmic Performance

The two–dimensional test case is flow past an axisymmetric submarine hull configuration known as *SUBOFF*. A (131 × 51 × 2) grid was used, which is highly stretched to resolve the very thin shear layer occuring at the high Reynolds number (12M based on hull length) for this case. For this grid, the ratio of maximum to minimum mesh spacing is 1.6M for the radial direction and 90 for the circumferential direction. The maximum cell aspect ratio is 12,778. The pressure and wall friction coefficients from the converged solutions for one, four, eight and twelve processors were compared and are in good agreement both with each other and with experimental data.

The most important algorithm parameters are the nondimensional time step (CFL number, defined in [6]) and the number of subiterations (I_S). The frequency of updating flux Jacobian linearizations was found to have only a weak influence on stability and iterative convergence rate. The number of iterations required to reduce the maximum residual by a factor of 10^{-3} are plotted in Figure 3 as a

function of CFL and I_S. The results in Figure 3 are re–plotted in Figure 4 to convert the number of iterations to arithmetic complexity using the CPU requirements from the performance estimate given in Section 4 for this solution algorithm. Note that the minimum arithmetic complexity occurs for I_S of 3 to 5 and CFL of about 40 to 50, indicating that further subiteration is less efficient in CPU time even though fewer time step iterations are required. The optimal CFL number is expected to vary somewhat for other flow problems.

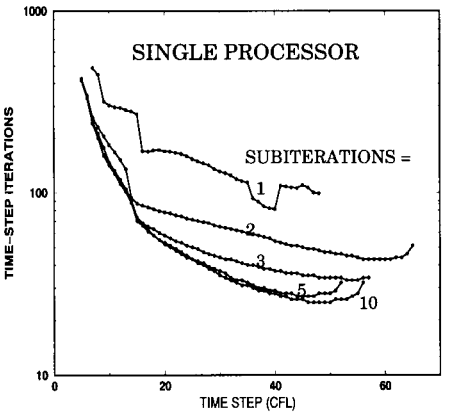


Figure 3. Iterations for *0(3)* Residual Reduction

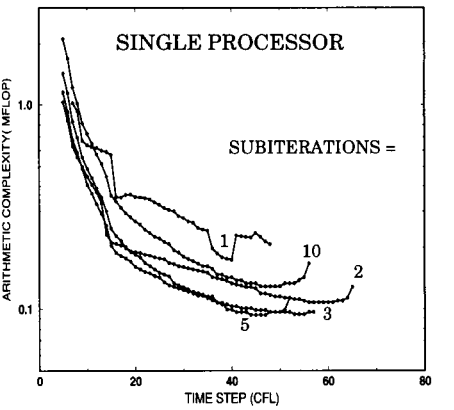


Figure 4. Complexity for *0(3)* Residual Reduction

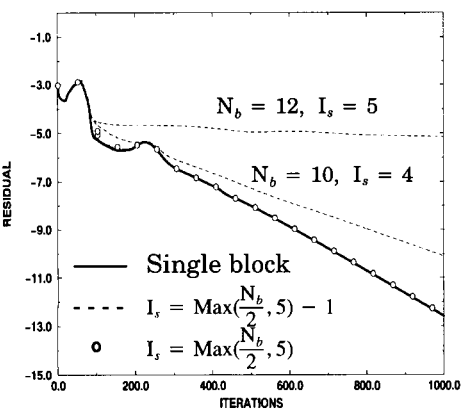


Figure 5. Effect of subiterations on convergence

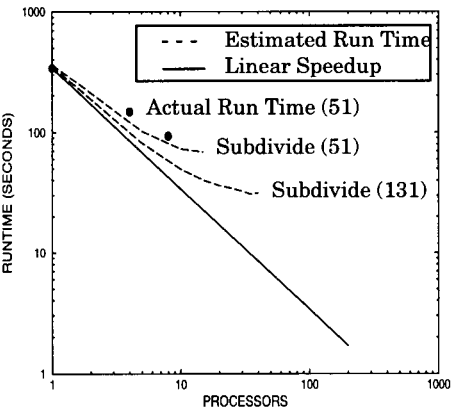


Figure 6. Run–time for 500 steps of SUBOFF on 4/8 nodes of the IBM SP2

Figure 5 gives results which suggest guidelines for selecting the number of subiterations for the decoupled parallel algorithm to maintain the convergence rate of the sequential algorithm. In a series of cases with different numbers of processors and subiterations and with one– and two–dimensional decompositions, it appears that the sequential convergence rate is maintained provided there is one subiteration for every two grid blocks in the direction of the longest block decomposition. This implies that a balanced higher dimensional decomposition will show the least amount of algorithmic degradation. It is significant that the

additional subiterations required are inexpensive for this algorithm because flux Jacobians are saved and reused.

5.3. Parallel Performance

The parallel performance estimates of Section 4 are compared in Figure 6 with actual run times for these one, four and eight processor cases on an IBM SP–2, and in general, the runtime estimates are in reasonable agreement with the actual trend. The reason for dividing the shorter direction (having 51 points) is that this direction cuts across the turbulent boundary layer. This demonstrates the ability of this particular algorithm to handle arbitrary domain decompositions.

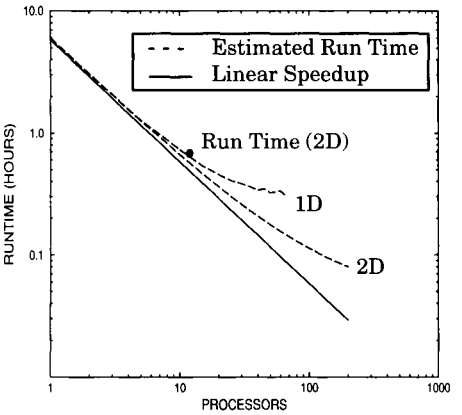


Figure 7. Run time for 500 steps of 0.6M case on 12 nodes of the IBM SP2

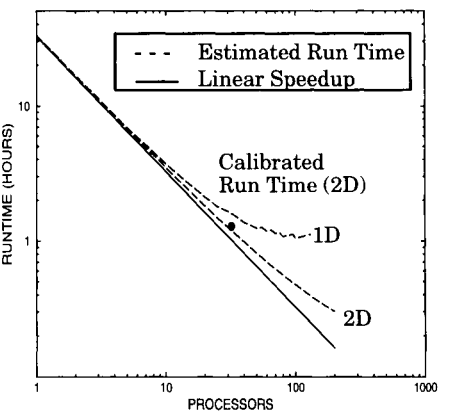


Figure 8. Run–time for 500 steps of 3.3M case on 32 nodes of the IBM SP2

In Figs. [7–8], estimated and actual/calibrated runtimes are compared for much larger cases for flow past a submarine hull with sail and stern appendages, having grids of 12 blocks of (49x41x25) (i.e., 0.6M points) and for 32 blocks of (49x65x33) (i.e., 3.3M points), with two–dimensional decompositions. Estimated times are given for both one– and two–dimensional decompositions. Calibrated timings are obtained by scaling 5 step runs. It can be seen from Figs. [7–8] that reasonable parallel performance can be obtained for these larger problems provided the grid and decomposition are appropriately sized for the targeted computer architecture. The 12 block and 32 block cases ran at effective speeds of 480Mflops and 1.25Gflops respectively.

6. CONCLUSIONS

(1) The decoupled subiteration algorithm used here allows the convergence rate of the sequential algorithm to be recovered at reasonable cost by using a sufficient number of subiterations, for the coarse–grained domain decomposition considered. As the problem size increases, algorithmic degradation can be avoided by resorting to balanced higher dimensional decompositions. The parallel algorithm also allows decompositions that are subdivided across boundary layers and is insensitive to various orderings of the sweep directions during subiteration.

(2) The parallel performance estimates obtained here for the IBM SP–2 are in reasonable agreement with actual run times for the test problem considered, in part because an accurate estimate of processor speed for the actual problem was obtained. These performance estimates seem generally useful for *a priori* problem definition even with approximate processor speeds.

(3) The present approach has given reasonable parallel performance for the larger scale steady flow simulations wherein the grid and decomposition were appropriately sized for the targeted computer architecture. Further work is needed to evaluate the scalability of this algorithm.

ACKNOWLEDGEMENTS

This research was supported by the Office of Naval Research (ONR) under grants N00039–92–C–0100 and N00014–92–J–1060, with J. A. Fein as the technical monitor. This work is part of a cooperative effort with Applied Research Laboratory at Pennsylvania State University. The authors also wish to express their sincere appreciation to Drs. L. K. Taylor, C. Sheng , A. Arabshahi and D. L. Whitfield for assistance with the submarine flow cases; to Dr. M.–Y. Jiang for generating the grids used in this work; to Dr. A. Skjellum and N. Doss for assistance in using MPI; and to H. Franke for help with MPI–F. The authors also gratefully acknowledge use of facilities at the Maui High–Performance Computing Center, the Mississippi Center for Supercomputing Research and the Argonne High–Performance Computing Research Facility.

REFERENCES

1. L. K. Taylor and D. L. Whitfield: Unsteady Three–Dimensional Incompressible Euler and Navier–Stokes Solver for Stationary and Dynamic Grids, AIAA Paper No. 91–1650, (1991).
2. D. L. Whitfield and L. K. Taylor: Discretized Newton–Relaxation Solution of High Resolution Flux–Difference Split Schemes, AIAA Paper No. 91–1539, (1991).
3. L. K. Taylor, A. Arabshahi and D. L. Whitfield: Unsteady Three–Dimensional Incompressible Navier–Stokes Computations for a Prolate Spheroid Undergoing Time–Dependent Maneuvers, AIAA Paper No. 95–0313, (1995).
4. C. Sheng, L. K. Taylor and D. L. Whitfield: Multiblock Multigrid Solution of Three–Dimensional Incompressible Turbulent Flows About Appended Submarine Configurations, AIAA Paper No. 95–0203, (1995).
5. *MPI: A Message–Passing Interface Standard*, Comp. Sci. Dept. Tech. Report CS–94–230, University of Tennessee, Knoxville, Tennessee, (1994).
6. W. R. Briley, S. S. Neerarambam and D. L. Whitfield: Implicit Lower–Upper/ Approximate–Factorization Algorithms for Viscous Incompressible Flows, AIAA Paper No. 95–1742–CP, (1995).
7. P.L. Roe, *J. Computational Physics*, 43 (1981) 357.
8. B. van Leer, *J. Computational Physics*, 32 (1979) 101.
9. W. K. Anderson, J. T. Thomas and B. van Leer: *AIAA Journal*, 24 (1986) 1453.
10. B.S. Baldwin, and H. Lomax: Thin–Layer Approximation and Algebraic Model for Separated Turbulent Flows, AIAA Paper No. 78–0257, (1978).
11. W. Gropp, E. Lusk and A. Skjellum: Using MPI, MIT Press, (1994) 185–188.
12. B. Saphir and S. Fineberg: http://lovelace.nas.nasa.gov/Parallel/SP2/MPIPerf/ report.html.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

Comparing the performance of multigrid and conjugate gradient algorithms on the Cray T3D

Y. F. Hu, D. R. Emerson and R. J. Blake

Daresbury Laboratory, CLRC, Warrington WA4 4AD, United Kingdom

On many distributed memory systems, such as the Intel iPSC/860, the multigrid algorithm suffers from extensive communication requirements, particularly on the coarse grid level, and is thus not very competitive in comparison to the conjugate gradient algorithm. These two algorithms are compared on the new Cray T3D MPP system for solving very large systems of linear equations (resulting from grids of the order 256^3 cells). It is found that the T3D has very fast communication and, most importantly, a low latency. As a result, the multigrid algorithm is found to be competitive with the conjugate gradient algorithm on the Cray T3D for solving very large linear systems from direct numerical simulation (DNS) of combustion. Results are compared to those obtained on the Intel iPSC/860.

1. INTRODUCTION

Two of the most popular iterative solvers involved in the solution of large scale linear systems are the conjugate gradient algorithm and the multigrid algorithm (combined with an appropriate smoothing method).

The multigrid algorithm is typically used with a stationary-type solver and is frequently the method of choice for sequential machines. In this approach, the computational grid is divided into a number of levels. For practical engineering problems the flow field is turbulent and this necessitates that the calculation will start on the finest grid. The residual, which indicates the extent to which a problem has converged, will reduce quite rapidly at first as the high frequency errors are damped out. However, to further reduce the residual by a given factor requires an increasing number of iterations. By transferring the problem to a coarser mesh, the low frequency errors that were present on the finer mesh now appear as high frequency errors and can, therefore, be readily reduced. Although a significant number of iterations may be required on the coarse meshes, each iteration only involves a relatively small amount of computation in comparison to the fine mesh and, for sequential problems, the overall workload can be substantially reduced. An alternative approach is to employ a nonstationary algorithm, such as the conjugate gradient algorithm. In this approach, the system of equations are written in the form $Ax = b$ and a global search algorithm, which gives the conjugate directions, is sought. A critical element in the success of conjugate gradient algorithms is forming an appropriate preconditioner.

To a large extent, these issues are well known and understood for sequential applications. However, their parallel implementation introduces some problematic areas and it is not always the case that the "best" sequential algorithm is the "best" parallel algorithm. This paper will therefore discuss the issues involved in developing efficient parallel versions of these algorithms and compare the performance of the two algorithms.

2. BACKGROUND AND MOTIVATION

The target application for the present study is the Direct Numerical Simulation (DNS) of turbulent combustion. This problem involves the solution of the three dimensional Navier-Stokes equations and a scalar transport equation that represents the reaction progress variable [1]. In all, there are six partial differential equations to be solved. A low Mach number approximation is made to allow the pressure to be decoupled from the density. However, this approximation fails to satisfy continuity and it is necessary to solve a resulting Poisson equation to enforce a divergence free solution. This equation set can be written as:

$$\nabla^2 P = \sigma \nabla \cdot u \tag{1}$$

where σ is a constant which depends on the time step and grid size and P and u represent the pressure and velocity field, respectively. At present, the problem is limited to a back-to-back flame in a cube in space. The boundary conditions are therefore periodic.

The matrix resulting from the solution to equation (1) is large, sparse and, for this particular case, symmetric. For this problem, the matrix has a regular structure and, with the periodic boundary conditions, it is also singular. The solution of the Poisson equation is the most time consuming part of the computation. Indeed, for very large problems ($\approx 300^3 - 400^3$), the solution takes 98% of the computational time. In an ideal situation a direct solver would be used. However, parallel direct solvers are not practicable for this size of problem as the fill-in required would increase the storage significantly. Another possibility would be to employ a Fast Fourier Transform (FFT) based algorithm. However, this would impose restrictions on the partitioning strategy and on implementing, in future, more complicated geometry and non-periodic boundary conditions. Iterative schemes are therefore the preferred method for solving such systems because of their low storage requirement. As previously described, the optimal solvers currently available are multigrid and conjugate gradient methods.

The parallel code was initially developed to run on a 64 node Intel iPSC/860 hypercube at Daresbury Laboratory. A standard grid partitioning strategy was employed which utilised the Single Process Multiple Data (SPMD) style of programming. The code was subsequently modified to run on the Cray T3D. This machine has 320 DEC Alpha 21064 processors but, as it is only possible to use partitions employing 2^n processors, the maximum configuration available is 256. The DEC Alpha chip operates at 150 MHz and therefore has a peak performance of 38.4 Gflop/s on 256 processors. The floating point performance of the machine has been discussed elsewhere ([1]) and, whilst this is a critical factor, it is necessary to establish the communication performance of the Cray T3D. For both conjugate gradient and multigrid to be efficient it is necessary to have a low latency and a high bandwidth. These figures were obtained by performing a *ping-pong* test whereby processor A sends a message to processor B and, once

the message has been received, processor B returns the message to A. This process is repeated for many times and the average time taken to do a send is then determined. It should be noted at this stage that the Cray T3D offers a wide range of message passing options involving PVM at the highest level, PVMFPSEND/PVMFPRECV at the next level, and the low level Cray routines SHMEM_PUT/GET. All of these routines were tested and the results on latency (t_0) and bandwidth (r_∞) are given in Table 1. The results in brackets for PVM and PVMFPSEND are for messages larger than 4096 Bytes. As can be seen from the table, the latency varies considerably with the message passing scheme being employed. Previous calculations on the Intel have shown that the multigrid algorithm was not as effective as the conjugate gradient algorithm because of the relatively high latency of the machine (70 μs [2]) and its low bandwidth (2.4 MBytes/s [2])

Table 1
Communication performance of the Cray T3D

	Uni-directional		Bi-directional	
Routine	t_0 (μs)	r_∞ (MBytes/s)	t_0 (μs)	r_∞ (MBytes/s)
PVM	138 (265)	26 (26)	-	-
PVMFPSEND	32 (222)	34 (26)	19 (148)	52 (48)
SHMEM_PUT	6	120	6	140
SHMEM_GET	6	60	-	-

The balance between the speed of communication and computation is also looked at. It is found that compared with the Intel i860, the Cray T3D is much more balanced. In fact because of the small cache and the lack of secondary cache, the computational speed is relatively slow compared with communication speed for large vectors. Thus for example when doing halo data transfer of large vectors, more time is spent in packing and unpacking than in communication itself. For small messages the communication is still slower compared with the computation because of the startup cost.

3. THE CONJUGATE GRADIENT AND MULTIGRID ALGORITHMS

As previously described, the matrix to be solved is symmetric, positive and semi-definite. This, therefore, makes it amenable to being solved by a standard conjugate gradient solver. The grid partitioning strategy allocates an $N \times N \times N$ grid into $N_x \times N_y \times N_z$ cells, where $N_x = N/p_x$ etc. and p_x represents the number of processors in the x-direction. The total number of processors employed is therefore $p = p_x \times p_y \times p_z$. The computational grid employed is staggered with the pressure being located at the cell centre and the velocities stored at the interfaces. The basic communication structure is illustrated in Figure 1 for a simple 2D slice. It should be noted that the staggered grid necessitates the transfer of data across the diagonal as indicated.

3.1 The conjugate gradient algorithm

The pseudo-code for the conjugate gradient algorithm can be written as follows:

$x_o = p_{-1} = \beta_{-1} = 0;\ r_o = b$
solve $M\omega_o = r_o$
$\rho_o = (r_o, \omega_o)$ (†)
For $i = 0, 1, 2, 3, \ldots$
$p_i = \omega_i + \beta_{i-1} p_{i-1}$ (†)
$q_i = A p_i$
$\alpha_i = \dfrac{\rho_i}{(p_i, q_i)}$ (†)
$x_{i+1} = x_i + \alpha_i p_i$ (†)
$r_{i+1} = r_i - \alpha_i q_i$ (†)
if $\|r_{i+1}\|_2 < \delta$ exit (†)
$\omega_{i+1} = M^{-1} r_{i+1}$
$\rho_{i+1} = (r_{i+1}, w_{i+1})$ (†)
$\beta_i = \dfrac{\rho_{i+1}}{\rho_i}$

where (,) represents the inner dot product of the vectors, δ is a specified tolerance and M a preconditioner.

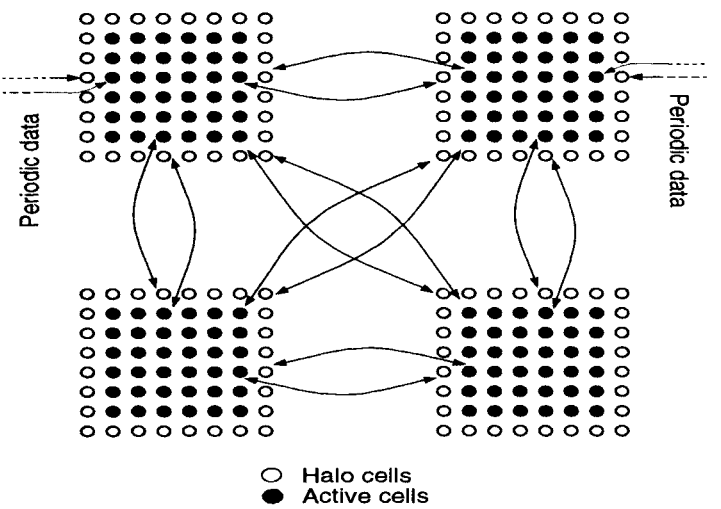


Figure 1 Basic communication strategy

There are several features within the pseudo-code that are worth noting: (i) The majority of the code can be written in Level 1 BLAS (e.g., the dot products can be written as *sdot*) and these routines have been indicated with the symbol (†); (ii) To evaluate a dot product requires the global summation of scalars. During the early phase of this project, the Cray T3D did not have an intrinsic routine to perform this operation

(unlike the Intel with *GDSUM*). The hand coded algorithm used to perform this operation was based on a hypercube summation pattern and all results quoted herein are using that algorithm. Subsequent tests with the Cray T3D's intrinsic summation routine have shown them to perform similarly for short vector lengths; (iii) The matrix-vector product ($q_i = Ap_i$) is performed with standard FORTRAN 77 and, due to the finite-difference stencil, it is necessary to transfer the halo (or ghost) cells. Another communication factor to be considered is the periodic boundary condition. This requires the transfer of data between processors that are not near neighbors. The matrix vector operation also accesses data that is not contiguous in memory and therefore places demands on the cache. To improve the performance of this section of the code, loop unrolling was performed on the Cray T3D. However, this was not found to be beneficial on the Intel. This requires the transfer of data between processors that are not near neighbours; (iv) The preconditioner employed was based on the Modified ILU (MILU) algorithm proposed by Chan and Kuo [3]. It is only performed on data local to the processor. This does incur a slight penalty whereby the number of iterations required to achieve a converged solution increase with the number of processors. However, for this problem, the increase is minimal and typically less than 5%. The preconditioning step also places significant demands on the cache. It should noted at this stage that the Cray T3D has no secondary cache and this clearly affects the performance of the matrix-vector product and the preconditioner. These issues, and the BLAS vs FORTRAN performance, are discussed in more detail by Emerson and Cant [1].

3.2 The multigrid algorithm

The are several variations based upon the multigrid algorithm. For the present investigation, the flow is turbulent and, as previously stated, it is necessary to begin on the finest grid. The method used for the present case is based upon that proposed by Hutchinson and Raithby [4]. In this approach, the coarse grid correction is formed using the *block correction* method and this has been chosen because of its flexibility. The pseudo-code for a 2-grid algorithm can be written as follows:

Iterate on the fine grid equation $A^f x = b^f$ to get x^f
Compute fine grid residual $r^f = b^f - A^f x^f$
Compute coarse grid residual $r^c = I_f^c r^f$
Form coarse grid equation $A^c x = r^c$
Iterate on $A^c x = r^c$ to get x^c
Correct the fine grid solution $x^f := x^f + I_c^f x^c$
Repeat the above process until converged

In the algorithm illustrated above I_f^c is the restriction operator to transfer from the fine to the coarse grid and I_c^f is the prolongation operator from the coarse to the fine grid.

With the block correction approach, the equations on the coarse grid cells are formed essentially by adding the equations for the fine grid cells. This make it easy to lump an arbitrary number of cells along any direction into a coarse grid cell, which is useful when the equation is anisotropic along certain direction. It also allows to cope with subdomains whose number of cells along a certain coordinate direction is not even,

which could happen when solving in parallel on a mesh that has been partitioned into many subdomains.

For each level of the grids, a few iterations are carried out on the grid using an appropriate parallel stationary iterative algorithm. Several algorithms have been implemented, including the Jacobi, the Gauss-Seidel (GS) and the ADI algorithms. It was found that, in general, multigrid coupled with GS takes less time to converge than with ADI, even though the latter may take less iterations. After each iteration, halo data are transferred and the residual is assessed (which incurs a global summation of scalars) to decide whether to continue solving on the current level, to solve for a correction on a coarser level (if the norm is not reducing faster than a given factor σ, we used $\sigma = 0.5$), or whether the current level has converged (if the residual is less than α times the initial residual, we used 1.0^{-6} for the first level and $\alpha = 0.1$ for all the other levels). If the given tolerance has been satisfied, the correction can be added to the finer level. It was found that on the coarsest grid, as the residual has to go down by the factor α, it was faster to solve the problem using the conjugate gradient algorithm with MILU preconditioner, rather than using the GS algorithm.

Multigrid algorithms tend to iterate the most on the coarsest grid. However, even though multigrid algorithms may need a large number of iterations to converge over all levels, the equivalent number of iterations on the finest grid is usually quite low because one coarse grid iteration only incurs a fraction of the computational cost of one finest grid iteration. Therefore, on a sequential machine, multigrid algorithms are very competitive with nonstationary iterative algorithms such as the conjugate gradient algorithm. However, on distributed memory machines, the large number of iterations on the coarse grids incur a large amount of communications with a short message length. As a result, multigrid algorithms were found to suffer significantly on many distributed memory parallel computers, such as the Intel iPSC/860, due to the high latency involved in sending small messages. With the much lower latency of the Cray T3D, the situation is improved, as will be seen from the next section.

4. NUMERICAL RESULTS

The conjugate gradient algorithm and the multigrid algorithm is used to solve the linear systems from DNS. The results of solving such a system associated with a mesh of size 64^3, using the conjugate gradient algorithm (CG) and the multigrid with Gauss-Siedel algorithm with 4 levels of grids (MGS4), on both Intel i860 and the Cray T3D, are compared in Table 2. The number of iterations, total elapsed time ($t_{elapsed}$, in seconds), time for halo data transfer (t_{trans}, in seconds) and time for global summation (t_{sum}, in seconds) are given. The computational domains are partioned into $4 \times 2 \times 2$ subdomains. On the Cray T3D, SHMEM_PUT routine is used for faster communication.

As can be seen from Table 2, even with only 16 processors, a lot of time is spent in the communication on the Intel iPSC/860, particularly for the multigrid algorithm. This is because there are a large number iterations on the coarse grid. Hence there are a lot of global summations and halo data transfers, and most with short message sizes. Since the Intel has a relatively high latency, this results in a very high communication overhead. Subsequently, the multigrid algorithm is slower than the conjugate gradient algorithm on the Intel, while on the Cray T3D the multigrid is faster.

Table 2
Solving a 64^3 problem on 16 processors: comparing the Cray T3D and the Intel i860

machine	algorithm	iterations	$t_{elapsed}$	t_{trans}	t_{sum}
Cray T3D	CG	144	11.3	0.4	0.01
Intel i860	CG	144	30.7	3.6	0.7
Cray T3D	MGS4	129.1	8.3	1.5	0.6
Intel i860	MGS4	129.1	42.0	15.7	9.2

The two algorithms are then used to solve a large problem associated with a grid size of 256^3 on the Cray T3D. The results are given in Table 3 and 4. The grids are partitioned into $8 \times 4 \times 4$ (for 128 processors) and $8 \times 8 \times 4$ (for 256 processors) subdomains respectively. Results using both PVMFPSEND (PVMFPRECV) and SHMEM are presented.

Table 3
Solving a 256^3 problem on 128 processors

machine	algorithm	iterations	$t_{elapsed}$	t_{trans}	t_{sum}
SHMEM	CG	474	243.5	5.9	0.2
SHMEM	MGS4	193.5	98.5	12.1	6.0
SHMEM	MGS5	173.0	91.4	14.4	9.5
PVM	CG	474	246.3	8.3	0.6
PVM	MGS4	193.5	122.2	21.2	20.4
PVM	MGS5	173.0	126.8	26.8	32.4

Table 4
Solving a 256^3 problem on 256 processors

machine	algorithm	iterations	$t_{elapsed}$	t_{trans}	t_{sum}
SHMEM	CG	443	116.3	3.4	0.2
SHMEM	MGS4	213.8	57.9	9.8	7.2
SHMEM	MGS5	193.0	61.0	13.2	12.6
PVM	CG	443	118.6	5.2	0.7
PVM	MGS4	213.8	82.5	17.6	23.9
PVM	MGS5	193.0	103.8	25.9	54.2

As can be seen from the tables, for this large linear system, the conjugate gradient algorithm takes a lot more iterations than when used for the 64^3 problem. In terms of the ratio between communication time and computation time, for the conjugate gradient algorithm the communication takes a very small percentage of the total time. This is because for the size of the problem solved, the subdomain on each processor is quite large (of size at least $34 \times 34 \times 64$), so the three global summations and one halo data transfer per iteration for the conjugate gradient algorithm do not take a significant percentage of the time. For the multigrid algorithm, because of the large amount of communication on the coarse grid, even though the T3D has very fast communication, the time spent in the global summation and halo data transfer are still very significant.

It is seen that sometimes it is beneficial to use less levels of grids to reduce the communication overhead associated with the coarse grids. For example, on 256 processors, MGS4 is faster than MGS5 due to the smaller communication times, even though the former takes more iterations and more computing than the latter.

5. CONCLUSION

Because of the good communication performance of the Cray T3D, the multigrid algorithm is found quite competitive against the conjugate gradient algorithm, even though the former has a communication overhead of over 30% on 256 processors. To improve the communication time for the multigrid, it is necessary to reduce the startup time of the communication (a main limiting factor for the global scalar summation) as well as to improve the packing / unpacking speed (a limiting factor for the transfer of halo data).

It is noted that to reduce the communication cost of the multigrid algorithm, it may be useful to solve the coarse grid problem on only one processor and distributed the result to the others (see, e.g., [5]). The preconditioning part of the conjugate gradient algorithm can also be improved by employing an approach suggested by Eisenstat [6].

Incidentally, during our numerical tests it was interesting to found that although multigrid was suggested to overcome the inability of the stationary type iterative algorithms in reducing the low frequency errors, in practice, the combination of multigrid with the preconditioned conjugate gradient algorithm on very large problems frequently converges faster than the preconditioned conjugate gradient algorithm itself, or the combination of the multigrid and a stationary iterative algorithm.

REFERENCES

1. D. R. Emerson and R. S. Cant, Direct simulation of turbulent combustion on the Cray T3D - initial thoughts and impressions from an engineering perspective, submitted to Parallel Computing.
2. J. Helin and R. Berrendorf, Analysing the performance of message passing hypercubes: a study with the Intel iPSC/860, 1991 ACM International Conference on Supercomputing, Cologne, Germany.
3. T. F. Chan and C.-C. J. Kuo, Parallel elliptic preconditioners: fourier analysis and performance on the Connection Machine, Computer Physics Communications, 53 (1989) 237-252.
4. B. R. Hutchinson and G. D. Raithby, A multigrid method based on additive correction strategy, Numerical Heat Transfer, 9 (1986) 511-537.
5. M. Alef, Concepts for efficient multigrid implementation on SUPRENUM-like architectures, Parallel Computing, 17 (1991) 1-16.
6. S. C. Eisenstat, Efficient implementation of a class of preconditioned conjugate gradient methods, SIAM Journal of Scientific and Statistical Computing, 2 (1981) 1-4.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Domain Decomposition/Fictitious Domain Methods with Nonmatching Grids for the Navier-Stokes Equations Parallel Implementation on a KSR1 Machine

Roland Glowinski[a,b], Tsorng-Whay Pan[a] and Jacques Periaux[c]

[a]Department of Mathematics, University of Houston, Houston, Texas 77204 USA

[b]Université P. et M. Curie, Paris, France

[c]Dassault Aviation, 92214 Saint-Cloud, France

In this paper we simulate incompressible viscous flow modelled by Navier-Stokes Equations on parallel MIMD machines with nonmatching grids. The method of choice is the one shot method based on the combination of fictitious domain and domain decomposition methods. A fine mesh is used in the neighborhood of the obstacle in order to capture the small scale phenomena taking place there and sufficiently far from the obstacle a coarse mesh can be used. However the interface compatibility conditions for nonmatching meshes has been imposed by well chosen Lagrange multipliers via a weak formulation. These methods have been applied to solve incompressible viscous flow problems at moderate Reynolds numbers around a disk. Numerical results and performances obtained on a KSR machine are presented.

1. INTRODUCTION

Fictitious domain methods for partial differential equations are showing interesting possibilities for solving complicated problems motivated by applications from science and engineering (see, for example, [1] and [2] for some impressive illustrations of the above statement). The main reason for the popularity of fictitious domain methods (sometimes called *domain embedding methods*; cf. [3]) is that they allow the use of fairly structured meshes on a simple shape auxiliary domain containing the actual one, therefore allowing the use of fast solvers.

In this article which follows [4-8], we consider the fictitious domain solution of the Navier-Stokes equations modelling the unsteady incompressible Newtonian viscous fluids on parallel MIMD machines with nonmatching grids. The method of choice is the one shot method based on the combination of fictitious domain and domain decomposition methods. A fine mesh is used in the neighborhood of the obstacle in order to capture the small scale phenomena taking place there. Sufficiently far from the obstacle a coarse mesh can be used and in order to avoid the use of a unique mesh with rapidly varying step size we prefer to introduce different regular meshes nonmatching at the interfaces. However the interface compatibility conditions can be imposed by well

chosen Lagrange multipliers via a weak formulation. The partition of domain and the implementation of method with nonmatching grids are straightforward.

Another advantage here is the fact that incompressibility, matching at the interfaces and boundary conditions are all treated by Lagrange multipliers whose set can be treated as a unique one. This approach is a much better method for parallel implementation than the nested iteration loop where the hierarchy is imposed to the multipliers as described in [4]. It also relies on the splitting methods described in, e.g., [9-11]; with these methods one can decouple the numerical treatments of the incompressibility and of the advection, and take advantage of this fact to use the embedding approach in the (linear) incompressibility step only, the advection being treated in the larger domain without concern -in some sense- for the actual boundary.

The content of this article is as follows: In Section 2 we consider fictitious domain methods for incompressible Navier-Stokes equations; then in Section 3 we discuss the iterative solution of the quasi-Stokes problems obtained by operator splitting methods. In Section 4 the above method is applied to simulate external incompressible viscous flow past a disc modelled by Navier-Stokes equations. Numerical results and performances obtained on a KSR machine are presented.

2. FORMULATIONS

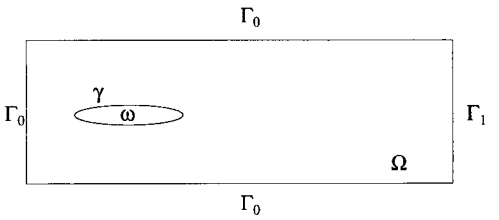


Figure 1.

Using the notation of Fig. 1, we consider the following problem

$$\frac{\partial \mathbf{u}}{\partial t} - \nu \Delta \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p = \mathbf{f} \ in \ \Omega \setminus \overline{\omega}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0 \ in \ \Omega \setminus \overline{\omega}, \tag{2}$$

$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0(\mathbf{x}), \quad \mathbf{x} \in \Omega \setminus \overline{\omega}, (with \ \nabla \cdot \mathbf{u}_0 = 0), \tag{3}$$

$$\mathbf{u} = \mathbf{g}_0(t) \, on \, \Gamma_0, \quad \mathbf{u} = \mathbf{g}_2(t) \, on \, \gamma, \quad \nu \frac{\partial \mathbf{u}}{\partial \mathbf{n}} - \mathbf{n}p = \mathbf{g}_1(t) \, on \, \Gamma_1. \tag{4}$$

In (1)-(4), Ω is bounded domains in $\mathbb{R}^d (d \geq 2)$, ω could be a moving rigid body in $\mathbb{R}^d (d \geq 2)$ (see Fig. 1), Γ (resp., γ) is the boundary of Ω (resp., ω) with $\Gamma = \Gamma_0 \cup \Gamma_1$, $\Gamma_0 \cap \Gamma_1 = \emptyset$, and $\int_{\Gamma_1} d\Gamma > 0$, $\mathbf{n}$ is the outer normal unit vector at Γ_1, $\mathbf{u} = \{u_i\}_{i=1}^{i=d}$ is the flow velocity, p is the pressure, $\mathbf{f}$ is a density of external forces, $\nu (> 0)$ is a viscosity parameter, and $(\mathbf{v} \cdot \nabla)\mathbf{w} = \{\sum_{j=1}^{j=d} v_j \frac{\partial w_i}{\partial x_j}\}_{i=1}^{i=d}$.

To obtain the equivalent fictitious domain formulation for the Navier-Stokes equations (see, e.g., [4,5]), we embed $\Omega \setminus \overline{\omega}$ in Ω and define $\mathbf{V}_{\mathbf{g}_0} = \{\mathbf{v} | \mathbf{v} \in (\mathbf{H}^1(\Omega))^d, \mathbf{v} = \mathbf{g}_0 \ on \ \Gamma_0\}$,

$\mathbf{V}_0 = \{\mathbf{v} | \mathbf{v} \in (\mathbf{H}^1(\Omega))^d, \mathbf{v} = \mathbf{0}\ on\ \Gamma_0\}$, and $\Lambda = (\mathbf{L}^2(\gamma))^d$. We observe that if $\mathbf{U}_0$ is an extension of $\mathbf{u}_0$ with $\nabla \cdot \mathbf{U}_0 = 0$ in Ω, and if $\tilde{\mathbf{f}}$ is an extension of $\mathbf{f}$, we have equivalence between (1)-(4) and the following problem:

$$\left\{ \begin{array}{l} For\ t > 0,\ find\ \mathbf{U}(t) \in \mathbf{V}_{\mathbf{g}_0},\ P(t) \in \mathbf{L}^2(\Omega),\ \lambda(t) \in \Lambda\ such\ that \\ \displaystyle\int_\Omega \frac{\partial \mathbf{U}}{\partial t} \cdot \mathbf{v}\, d\mathbf{x} + \nu \int_\Omega \nabla \mathbf{U} \cdot \nabla \mathbf{v}\, d\mathbf{x} + \int_\Omega (\mathbf{U} \cdot \nabla)\mathbf{U} \cdot \mathbf{v}\, d\mathbf{x} - \int_\Omega P \nabla \cdot \mathbf{v}\, d\mathbf{x} \\ \displaystyle = \int_\Omega \tilde{\mathbf{f}} \cdot \mathbf{v}\, d\mathbf{x} + \int_{\Gamma_1} \mathbf{g}_1 \cdot \mathbf{v}\, d\Gamma + \int_\gamma \lambda \cdot \mathbf{v}\, d\gamma,\ \forall \mathbf{v} \in \mathbf{V}_0,\ a.e.\ t > 0, \end{array} \right. \tag{5}$$

$$\nabla \cdot \mathbf{U}(t) = 0 \text{ in } \Omega, \quad \mathbf{U}(\mathbf{x}, 0) = \mathbf{U}_0(\mathbf{x}),\ \mathbf{x} \in \Omega, \text{ (with } \nabla \cdot \mathbf{U}_0 = 0), \tag{6}$$

$$\mathbf{U}(t) = \mathbf{g}_2(t) \text{ on } \gamma, \tag{7}$$

in the sense that $\mathbf{U}|_{\Omega \setminus \overline{\omega}} = \mathbf{u}$, $P|_{\Omega \setminus \overline{\omega}} = p$. Above, we have used the notation $\phi(t)$ for the function $\mathbf{x} \to \phi(\mathbf{x}, t)$.

Concerning the multiplier λ, its interpretation is simple since it is equal to the jump of $\nu(\partial \mathbf{U}/\partial \mathbf{n}) - \mathbf{n}P$ at γ. Closely related approaches are discussed in [12, 13]. We observe that the effect of the actual geometry is concentrated on $\int_\gamma \lambda \cdot \mathbf{v}\, d\gamma$ in the right-hand-side of (5), and on (7).

To solve (5)-(7), we shall consider a time discretization by an *operator splitting method*, like the ones discussed in, e.g., [9-11]. With these methods we are able to *decouple* the nonlinearity and the incompressibility in the Navier-Stokes/fictitious domain problems (5)-(7). Applying the *θ-scheme* (cf. [11]) to (5)-(7), we obtain two *quasi-Stokes/fictitious domain subproblems* and a nonlinear *advection-diffusion* subproblems at each time step (e.g., see [4, 5]). In Section 3, an one shot method for the quasi–Stokes/FD subproblems shall be discussed. Due to the fictitious domain method and the operator splitting method, advection–diffusion subproblems may be solved in a least–squares formulation by a conjugate gradient alogrithm [11] in a simple shape auxiliary domain Ω without concern for the constraint $\mathbf{u} = \mathbf{g}$ at γ. Thus, advection–diffusion subproblems can be solved with domain decomposition methods.

3. QUASI-STOKES PROBLEM AND ITS ITERATIVE SOLUTIONS

The quasi-Stokes/fictitious domain problem is the following

$$\left\{ \begin{array}{l} Find\ \mathbf{U} \in \mathbf{V}_{\mathbf{g}_0},\ P \in \mathbf{L}^2(\Omega),\ \lambda \in \Lambda\ such\ that \\ \displaystyle \alpha \int_\Omega \mathbf{U} \cdot \mathbf{v}\, d\mathbf{x} + \nu \int_\Omega \nabla \mathbf{U} \cdot \nabla \mathbf{v}\, d\mathbf{x} - \int_\Omega P \nabla \cdot \mathbf{v}\, d\mathbf{x} \\ \displaystyle = \int_\Omega \mathbf{f} \cdot \mathbf{v}\, d\mathbf{x} + \int_\gamma \lambda \cdot \mathbf{v}\, d\gamma + \int_{\Gamma_1} \mathbf{g}_1 \cdot \mathbf{v}\, d\Gamma,\ \forall \mathbf{v} \in \mathbf{V}_0, \end{array} \right. \tag{8}$$

$$\nabla \cdot \mathbf{U} = 0\ in\ \Omega, \quad \mathbf{U} = \mathbf{g}_2\ on\ \gamma, \tag{9}$$

where, in (8), $\alpha (> 0)$ is the reciprocal of a partial time step. In (8)-(9), P (resp., λ) appears to be a Lagrange multiplier associated with $\nabla \cdot \mathbf{U} = 0$ (resp., $\mathbf{U} = \mathbf{g}_2$ on γ).

We can solve the above saddle-point system (8)-(9) by a conjugate gradient algorithm called the *one shot method* (see, e.g., [5]) driven by the pressure P and the multiplier λ, simultaneously.

3.1. Domain decomposition approach

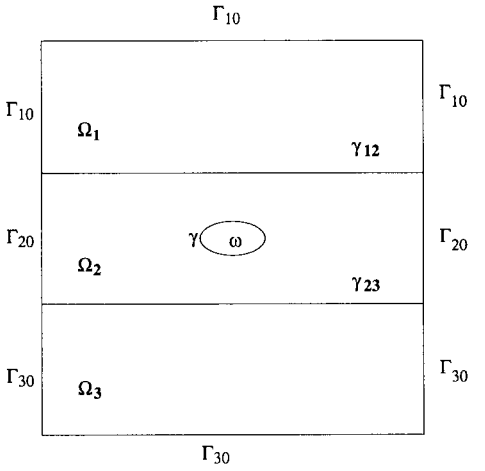


Figure 2.

To capture the small scale phenomena taking place in the neighborhood of the obstacle and avoid the use of a unique mesh with rapidly varying step size, we prefer to introduce different regular meshes through domain decomposition so that we can use fine mesh near obstacle and a coarser mesh sufficiently far from the obstacle. The compatibility conditions for these nonmatching meshes at the interfaces can be imposed by well chosen Lagrange multipliers via a weak formulation. For simplicity, let $\Omega = \Omega_1 \cup \Omega_2 \cup \Omega_3$ and $\Gamma_0 = \partial\Omega$ (see Fig. 2). We define spaces $V^i_{\mathbf{g}_0} = \{\mathbf{v} | \mathbf{v} \in (H^1(\Omega_i))^d, \mathbf{v} = \mathbf{g}_0 \text{ on } \Gamma_{i0}\}$, $V^i_0 = \{\mathbf{v} | \mathbf{v} \in (H^1(\Omega_i))^d, \mathbf{v} = \mathbf{0} \text{ on } \Gamma_{i0}\}$, for i =1, 2, 3; $\mathbf{V} = V^1_{\mathbf{g}_0} \times V^2_{\mathbf{g}_0} \times V^3_{\mathbf{g}_0}$, $\mathbf{V}_0 = V^1_0 \times V^2_0 \times V^3_0$, $\Lambda_p = \{(P_1, P_2, P_3) | P_i \in L^2(\Omega_i), i = 1, 2, 3, \ \sum_{i=1}^3 \int_{\Omega_i} P_i \, dx = 0\}$ and $\Lambda_\lambda = (L^2(\gamma))^d \times (L^2(\gamma_{12}))^d \times (L^2(\gamma_{23}))^d$. Problem (8)-(9) is equivalent to the following system:

Find $(\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3) \in \mathbf{V}$, $(P_1, P_2, P_3) \in \Lambda_p$, $(\lambda_b, \lambda_{12}, \lambda_{23}) \in \Lambda_\lambda$ *such that*

$$\sum_{i=1}^{3} \int_{\Omega_i} (\alpha \mathbf{U}_i \cdot \mathbf{v}_i + \nu \nabla \mathbf{U}_i \cdot \nabla \mathbf{v}_i - P_i \nabla \cdot \mathbf{v}_i) \, dx = \sum_{i=1}^{3} \int_{\Omega_i} \mathbf{f} \cdot \mathbf{v}_i \, dx \tag{10}$$

$$+ \int_\gamma \lambda_b \cdot \mathbf{v}_2 \, d\gamma + \int_{\gamma_{12}} \lambda_{12} \cdot (\mathbf{v}_2 - \mathbf{v}_1) \, d\gamma_{12} + \int_{\gamma_{23}} \lambda_{23} \cdot (\mathbf{v}_3 - \mathbf{v}_2) \, d\gamma_{23}, \quad \forall (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3) \in \mathbf{V}_0,$$

$$\nabla \cdot \mathbf{U}_i = 0 \text{ in } \Omega_i \quad \text{for } i = 1, 2, 3, \tag{11}$$

$$\int_\gamma \mu_b \cdot (\mathbf{U}_2 - \mathbf{g}_2) \, d\gamma = 0 \quad \forall \mu_b \in (L^2(\gamma))^d \tag{12}$$

$$\int_{\gamma_{12}} \mu_{12} \cdot (\mathbf{U}_2 - \mathbf{U}_1) \, d\gamma_{12} = 0 \quad \forall \mu_{12} \in (L^2(\gamma_{12}))^d \tag{13}$$

$$\int_{\gamma_{23}} \mu_{23} \cdot (\mathbf{U}_3 - \mathbf{U}_2) \, d\gamma_{23} = 0 \quad \forall \mu_{23} \in (L^2(\gamma_{23}))^d \tag{14}$$

We have equivalence in the sense that if relations (10)-(14) hold then $\mathbf{U}_i = \mathbf{U}|_{\Omega_i}$, for $i = 1, 2, 3$, where $\mathbf{U}$ is solution of (8)-(9), and conversely. There are *three* Lagrange multipliers in (10)-(14): (1) λ_b is the one associated with the *boundary condition* $\mathbf{U} = \mathbf{g}_0$ on γ; (2) λ_{12} (resp., γ_{23}) is the one associated with the *interface boundary condition* $\mathbf{U}_1 = \mathbf{U}_2$ (resp., $\mathbf{U}_2 = \mathbf{U}_3$) on γ_{12} (resp., γ_{23}); (3) pressure (P_1, P_2, P_3) is the one associated with (11).

We can also solve the above saddle-point system (10)-(14) by a conjugate gradient algorithm driven by *three* Lagrange multipliers, pressure (P_1, P_2, P_3), multipliers λ_b and $(\lambda_{12}, \lambda_{12})$, simultaneously as follows:

$$\{P^0, \lambda^0\} = \{(P_1^0, P_2^0, P_3^0), (\lambda_b^0, \lambda_{12}^0, \lambda_{23}^0)\} \in \Lambda_p \times \Lambda_\lambda \text{ given;} \tag{15}$$

solve the following elliptic problem:

Find $\mathbf{U}^0 = (\mathbf{U}_1^0, \mathbf{U}_2^0, \mathbf{U}_3^0) \in \mathbf{V}$, *such that*

$$\sum_{i=1}^{3} \int_{\Omega_i} (\alpha \mathbf{U}_i^0 \cdot \mathbf{v}_i + \nu \nabla \mathbf{U}_i^0 \cdot \nabla \mathbf{v}_i \, dx = \sum_{i=1}^{3} \int_{\Omega_i} \mathbf{f} \cdot \mathbf{v}_i \, dx + \sum_{i=1}^{3} \int_{\Omega_i} P_i \nabla \cdot \mathbf{v}_i \, dx \tag{16}$$

$$+ \int_\gamma \lambda_b^0 \cdot \mathbf{v}_2 \, d\gamma + \int_{\gamma_{12}} \lambda_{12}^0 \cdot (\mathbf{v}_2 - \mathbf{v}_1) \, d\gamma_{12} + \int_{\gamma_{23}} \lambda_{23}^0 \cdot (\mathbf{v}_3 - \mathbf{v}_2) \, d\gamma_{23} \quad \forall (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3) \in \mathbf{V}_0,$$

set

$$\mathbf{r}^0 = (r_{P,1}^0, r_{P,2}^0, r_{P,3}^0) = (\nabla \cdot \mathbf{U}_1^0, \nabla \cdot \mathbf{U}_2^0, \nabla \cdot \mathbf{U}_3^0), \tag{17}$$

$$\mathbf{g}_\lambda^0 = (\mathbf{g}_{\lambda_b}^0, \mathbf{g}_{\lambda_{12}}^0, \mathbf{g}_{\lambda_{23}}^0) = \left((\mathbf{U}_2^0 - \mathbf{g}_2)|_\gamma, (\mathbf{U}_2^0 - \mathbf{U}_1^0)|_{\gamma_{12}}, (\mathbf{U}_3^0 - \mathbf{U}_2^0)|_{\gamma_{23}}\right), \tag{18}$$

and define $\mathbf{g}^0 = (\mathbf{g}_P^0, \mathbf{g}_\lambda^0)$ *where* $\mathbf{g}_P^0 = (g_{P,1}^0, g_{P,2}^0, g_{P,3}^0)$ *is as follows:*

$$g_{P,i}^0 = \alpha \phi_i^0 + \nu r_{P,i}^0, \text{ for } i = 1, 2, 3, \tag{19}$$

with ϕ_i^0, *for* $i =$ *1,2,3, the solution of*

$$\begin{cases} -\Delta \phi_i^0 = r_{P,i}^0 \text{ in } \Omega_i, \\ \dfrac{\partial \phi_i^0}{\partial \mathbf{n}} = 0 \text{ on } \Gamma_{i0}; \ \phi_i^0 = 0 \text{ on } \Gamma_{i1}, \end{cases} \tag{20}$$

where $\Gamma_{11} = \gamma_{12}$, $\Gamma_{21} = \gamma_{12} \cup \gamma_{23}$ *and* $\Gamma_{31} = \gamma_{23}$. *We take*

$$\mathbf{w}^0 = (\mathbf{w}_P^0, \mathbf{w}_\lambda^0) = \{(w_{P,1}^0, w_{P,2}^0, w_{P,3}^0), (\mathbf{w}_{\lambda_b}^0, \mathbf{w}_{\lambda_{12}}^0, \mathbf{w}_{\lambda_{23}}^0)\} \tag{21}$$

$$= \{(g_{P,1}^0, g_{P,2}^0, g_{P,3}^0), (\mathbf{g}_{\lambda_b}^0, \mathbf{g}_{\lambda_{12}}^0, \mathbf{g}_{\lambda_{23}}^0)\}.$$

Then for $n \geq 0$, *assuming that* P^n, λ^n, $\mathbf{U}^n$, $\mathbf{r}^n$, $\mathbf{w}^n$, $\mathbf{g}^n$ *are known, compute* P^{n+1}, λ^{n+1}, $\mathbf{U}^{n+1}$, $\mathbf{r}^{n+1}$, $\mathbf{w}^{n+1}$, $\mathbf{g}^{n+1}$ *as follows:*

solve the intermediate elliptic problem:

Find $\bar{\mathbf{U}}^n = (\bar{\mathbf{U}}_1^n, \bar{\mathbf{U}}_2^n, \bar{\mathbf{U}}_3^n) \in \mathbf{V}_0$, *such that*

$$\sum_{i=1}^{3} \int_{\Omega_i} (\alpha \bar{\mathbf{U}}_i^n \cdot \mathbf{v}_i + \nu \nabla \bar{\mathbf{U}}_i^n \cdot \nabla \mathbf{v}_i \, dx = \sum_{i=1}^{3} \int_{\Omega_i} w_{P,i}^n \nabla \cdot \mathbf{v}_i \, dx + \int_\gamma \mathbf{w}_{\lambda_b}^n \cdot \mathbf{v}_2 \, d\gamma \tag{22}$$

$$+ \int_{\gamma_{12}} \mathbf{w}_{\lambda_{12}}^n \cdot (\mathbf{v}_2 - \mathbf{v}_1) \, d\gamma_{12} + \int_{\gamma_{23}} \mathbf{w}_{\lambda_{23}}^n \cdot (\mathbf{v}_3 - \mathbf{v}_2) \, d\gamma_{23} \quad \forall (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3) \in \mathbf{V}_0,$$

set $\bar{\mathbf{r}}^n = (\bar{r}^n_{P,1}, \bar{r}^n_{P,2}, \bar{r}^n_{P,3}) = (\nabla \cdot \bar{\mathbf{U}}^n_1, \nabla \cdot \bar{\mathbf{U}}^n_2, \nabla \cdot \bar{\mathbf{U}}^n_3)$, (23)

$$\bar{\mathbf{g}}^n_\lambda = (\bar{\mathbf{g}}^n_{\lambda_b}, \bar{\mathbf{g}}^n_{\lambda_{12}}, \bar{\mathbf{g}}^n_{\lambda_{23}}) = \left(\bar{\mathbf{U}}^n_2|_\gamma, (\bar{\mathbf{U}}^n_2 - \bar{\mathbf{U}}^n_1)|_{\gamma_{12}}, (\bar{\mathbf{U}}^n_3 - \bar{\mathbf{U}}^n_2)|_{\gamma_{23}}\right), \tag{24}$$

and define $\bar{\mathbf{g}}^n = (\bar{\mathbf{g}}^n_P, \bar{\mathbf{g}}^n_\lambda)$ *where* $\bar{\mathbf{g}}^n_P = (\{\bar{g}^n_{P,1}, \bar{g}^n_{P,2}, \bar{g}^n_{P,3})$ *is as follows:*

$$\bar{g}^n_{P,i} = \alpha \bar{\phi}^n_i + \nu \bar{r}^n_{P,i}, \text{ for } i = 1, 2, 3, \tag{25}$$

with $\bar{\phi}^n_i$, *for* $i =$ *1,2,3, the solution of*

$$\begin{cases} -\Delta \bar{\phi}^n_i = \bar{r}^n_{P,i} \text{ in } \Omega_i, \\ \dfrac{\partial \bar{\phi}^n_i}{\partial \mathbf{n}} = 0 \text{ on } \Gamma_{i0};\ \bar{\phi}^n_i = 0 \text{ on } \Gamma_{i1}, \end{cases} \tag{26}$$

where $\Gamma_{11} = \gamma_{12}$, $\Gamma_{21} = \gamma_{12} \cup \gamma_{23}$ *and* $\Gamma_{31} = \gamma_{23}$. *We compute then*

$$\rho_n = \Big(\sum_{i=1}^3 \int_{\Omega_i} r^n_{P,i} g^n_{P,i}\, d\mathbf{x} + \|\mathbf{g}^n_\lambda\|^2\Big) / \Big(\sum_{i=1}^3 \int_{\Omega_i} \bar{r}^n_{P,i} w^n_{P,i}\, d\mathbf{x} + < \bar{\mathbf{g}}_{\lambda_b}, \mathbf{w}^n_\lambda >\Big),$$

and set $P^{n+1}_i = P^n_i - \rho_n \mathbf{w}^n_P$, $\lambda^{n+1} = \lambda^n - \rho_n \mathbf{w}^n_\lambda$, $\mathbf{U}^{n+1} = \mathbf{U}^n - \rho_n \bar{\mathbf{U}}^n$, (27)

$$\mathbf{r}^{n+1} = \mathbf{r}^n - \rho_n \bar{\mathbf{r}}^n,\ \mathbf{g}^{n+1}_P = \mathbf{g}^n_P - \rho_n \bar{\mathbf{g}}^n_P,\ \mathbf{g}^{n+1}_\lambda = \mathbf{g}^n_\lambda - \rho_n \bar{\mathbf{g}}^n_\lambda. \tag{28}$$

If $\Big(\sum_{i=1}^3 \int_{\Omega_i} r^{n+1}_{P,i} g^{n+1}_{P,i}\, d\mathbf{x} + \|\mathbf{g}^{n+1}_\lambda\|^2\Big) / \Big(\sum_{i=1}^3 \int_{\Omega_i} r^0_{P,i} g^0_{P,i}\, d\mathbf{x} + \|\mathbf{g}^0_\lambda\|^2\Big) \le \epsilon$, *take* $\lambda = \lambda^{n+1}$, $P = P^{n+1}$ *and* $\mathbf{U} = \mathbf{U}^{n+1}$. *If not, compute*

$$\gamma_n = \Big(\sum_{i=1}^3 \int_{\Omega_i} r^{n+1}_{P,i} g^{n+1}_{P,i}\, d\mathbf{x} + \|\mathbf{g}^{n+1}_\lambda\|^2\Big) / \Big(\sum_{i=1}^3 \int_{\Omega_i} r^n_{P,i} g^n_{P,i}\, d\mathbf{x} + \|\mathbf{g}^n_\lambda\|^2\Big),$$

and set

$$\mathbf{w}^{n+1} = \mathbf{g}^{n+1} + \gamma_n \mathbf{w}^n. \tag{29}$$

Do $n = n + 1$ *and go back to* (22).

In (15)-(29), $< \mathbf{g}_\lambda, \mathbf{w}_\lambda >= \int_\gamma \mathbf{g}_{\lambda_b} \cdot \mathbf{w}_{\lambda_b}\, d\gamma + \int_{\gamma_{12}} \mathbf{g}_{\lambda_{12}} \cdot \mathbf{w}_{\lambda_{12}}\, d\gamma + \int_{\gamma_{23}} \mathbf{g}_{\lambda_{23}} \cdot \mathbf{w}_{\lambda_{23}}\, d\gamma$ and $\|\mathbf{g}_\lambda\|^2 =< \mathbf{g}_\lambda, \mathbf{g}_\lambda >$ for $\mathbf{g}_\lambda, \mathbf{w}_\lambda \in \Lambda_\lambda$. The elliptic problems (16), (20), (22) and (26) in the one shot method (15)-(29) can be solved simultaneously on MIMD machines.

4. Numerical experiments

In the test problem, $\Omega = (-0.75, 0.75) \times (-0.5, 0.5)$ and ω is a disk of radius 0.05 centered at the origin (see Fig. 3). The boundary conditions in (4) are $\mathbf{g}_0 = ((1 - e^{-ct}), 0)$ on Γ_0, $\mathbf{g}_2 = \mathbf{0}$ on γ and $\mathbf{g}_1 = \mathbf{0}$ on Γ_1. Ω is divided into five subdomains, $\{\Omega_i\}_{i=1}^5$, (see Fig. 3) and γ_{ij} denotes the interface between Ω_i and Ω_j. For finite dimensional subspaces on Ω_i, we choose

$$V^i_{\mathbf{g}_0,h} = \{\mathbf{v}_h | \mathbf{v}_h \in H^1_{g^1_0,h} \times H^i_{g^2_0,h}\},\ V^i_{0,h} = \{\mathbf{v}_h | \mathbf{v}_h \in H^i_{0,h} \times H^i_{0,h}\},$$

$$H^i_{g^j_0,h} = \{\phi_h | \phi_h \in C^0(\bar{\Omega}_i), \phi_h|_T \in P_1,\ \forall T \in \mathcal{T}^i_h,\ \phi_h = g^j_0 \text{ on } \Gamma_{i0}\},$$

$$H^i_{0,h} = \{\phi_h | \phi_h \in C^0(\bar{\Omega}_i), \phi_h|_T \in P_1,\ \forall T \in \mathcal{T}^i_h,\ \phi_h = 0 \text{ on } \Gamma_{i0}\},$$

where $\mathcal{T}_h^i$ is a triangulation of Ω_i (see, e.g, Figure 3), P_1 being the space of the polynomials in x_1, x_2 of degree ≤ 1. Finite dimensional subspace for pressure is $H_{2h}^i = \{\phi_h | \phi_h \in C^0(\bar{\Omega}_i), \phi_h|_T \in P_1, \ \forall T \in \mathcal{T}_{2h}^i\}$ where $\mathcal{T}_{2h}^i$ is a triangulation twice coarser than $\mathcal{T}_h^i$. The finite dimensional subspace Λ_h of each multiplier space $(L^2(\gamma))^2$ (here γ is either $\partial\omega$ or interface γ_{ij}) is

$$\Lambda_h = \{\mu_h | \mu_h \in (L^\infty(\gamma))^2, \mu_h \textit{ is constant on the segment joining } 2 \textit{ consecutive mesh points on } \gamma\},$$

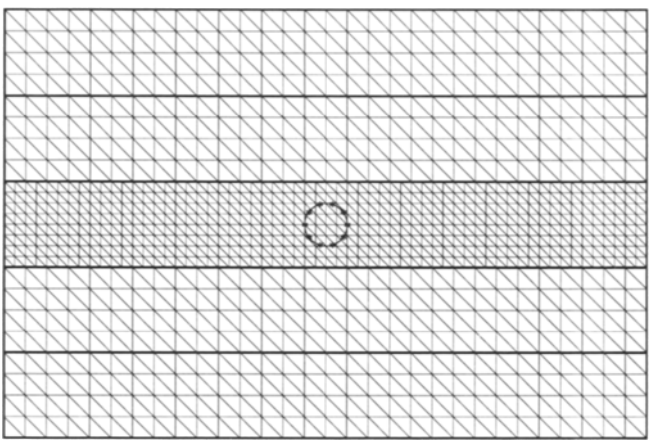


Figure 3. Mesh points marked by "$*$" on γ and the triangulation of $\Omega = \cup_{i=1}^5 \Omega_i$ with meshsizes $h = 1/20$ and $h = 1/40$.

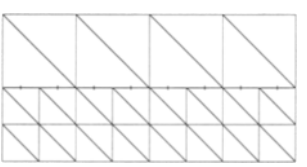


Figure 4. Example of mesh on interface between subdomains with nonmatching grids. Mesh points on interface are marked by "|"

In numerical simulation, the meshsize for velocity (resp., pressure) is $h_v = 1/100$ (resp., $h_p = 1/50$) for coarse grid in subdomains except the middle one. For fine grid in the middle subdomain, the meshsize for velocity (resp., pressure) is $h_v = 1/200$ (resp., $h_p = 1/100$). Time step is $\triangle t = 0.0015$ and $\nu = 0.001$; thus Reynolds number is 100 (taking the diameter of disk as characteristic length). For the stopping criterion in (18)-(29) ϵ is 10^{-12}. As running in KSR1 with 3 processors, the speedup is 1.66. The one shot method appears to be well suited to parallel computation. We would like now to simulate incompressible viscous flow modelled by Navier-Stokes equations on other parallel architectures and develop efficient preconditioners for it.

ACKNOWLEDGEMENT

We would like to acknowledge the helpful comments and suggestions of E. J. Dean, J.W. He, and J. Singer and the support of the following corporations and institutions: AWARE, Dassault Aviation, TCAMC, University of Houston, Université P. et M. Curie. We also benefited from the support of DARPA (Contracts AFOSR F49620-89-C-0125 and AFOSR-90-0334), DRET (GRANT 89424),NSF (GRANTS INT 8612680, DMS 8822522 and DMS 9112847) and the Texas Board of Higher Education (Grant 003652156ARP and 003652146ATP).

REFERENCES

1. J. E. Bussoletti, F. T. Johnson, S. S. Samanth, D. P. Young, R. H. Burkhart, EM-TRANAIR: Steps toward solution of general 3D Maxwell's equations, in Computer Methods in Applied Sciences and Engineering, R. Glowinski, ed., Nova Science, Commack, NY (1991) 49.
2. D. P. Young, R. G. Melvin, M. B. Bieterman, F. T. Johnson, S. S. Samanth, J. E. Bussoletti, A locally refined finite rectangular grid finite element method. Application to Computational Physics, J. Comp. Physics, **92** (1991) 1.
3. B. L. Buzbee, F. W. Dorr, J. A. George, G. H. Golub, The direct solution of the discrete Poisson equation on irregular regions, SIAM J. Num. Anal., **8** (1971) 722.
4. R. Glowinski, T. W. Pan, J. Periaux, A fictitious domain method for external incompressible viscous flow modeled by Navier-Stokes equations, Comp. Meth. Appl. Mech. Eng., **112** (1994) 133.
5. R. Glowinski, T. W. Pan, J. Periaux, A Lagrange multiplier/fictitious domain method for the Dirichlet problem. Generalization to some flow problems, Japan J. of Industrial and Applied Mathematics, **12** (1995) 87.
6. R. Glowinski, A. J. Kearsley, T. W. Pan, J. Periaux, Fictitious domain methods for viscous flow simulation, CFD Review (1995) (to appear).
7. R. Glowinski, T. W. Pan, J. Periaux, A one shot domain decomposition/fictitious domain method for the solution of elliptic equations, in Parallel C.F.D.: New Trends and Advances, A. Ecer, J. Hauser, P. Leca and J. Periaux eds., North–Holland, Amsterdam (1995) 317.
8. R. Glowinski, T. W. Pan, J. Periaux, One shot fictitious domain/domain decomposition methods for three-dimensional elliptic problems. Parallel implementation on a KSR1 machine, in the proceeding of Parallel CFD'94 (to appear).
9. R. Glowinski, Viscous flow simulation by finite element methods and related numerical techniques, Progress and Supercomputing in Computational Fluid Dynamics, E. M. Murman and S. S. Abarbanel eds., Birkhauser, Boston (1985) 173.
10. M.O. Bristeau, R. Glowinski and J. Periaux, Numerical methods for the Navier-Stokes equations, Comp. Phys. Rep., **6** (1987) 73.
11. R. Glowinski, Finite element methods for the numerical simulation of incompressible viscous flow. Introduction to the control of the Navier-Stokes equations, Lectures in Applied Mathematics,, AMS, Providence, RI, **28** (1991) 219.
12. C. Borgers, Domain embedding methods for the Stokes equations, Num. Math., **57** (1990) 435.
13. F. Bertrand, P.A. Tanguy, F. Thibault, A three-dimensional fictitious domain method for incompressible flow problems in enclosures containing moving parts (to appear).

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A Parallel Implicit Time Accurate Navier-Stokes Solver

Carl B. Jenssen and Karstein Sørli [a]

[a]SINTEF Industrial Mathematics, N-7034 Trondheim, Norway

An implicit multiblock method is used to obtain the solution of the time dependent Navier-Stokes equations. Approximate Newton iterations are used at each time step to invert the implicit operator. To ensure global coupling and fast converge in a multiblock framework, a global coarse grid correction scheme is applied together with explicit block interface conditions.

1. INTRODUCTION

Time accuracy complicates the use of an implicit scheme in a multiblock framework. Whereas only the accuracy of the converged solution is of interest for steady state calculations, and violation of the block interface conditions can be permitted during the transient phase, global accuracy must be ensured at each time step for time dependent computations. In the same way as convergence to steady state might slow down as the number of blocks is increased, the work needed per time step for a given level of accuracy might increase for time dependent problems.

In this paper we will investigate the use of a Coarse Grid Correction Scheme (CGCS) for obtaining time accurate solutions to the Navier-Stokes equations. This method has proven to be highly efficient for steady state computations [3], and is here extended to time dependent problems on moving meshes.

The flow solver is written for a general three-dimensional structured multiblock mesh, allowing for both complex geometries and parallel computations [1,2]. The compressible Navier-Stokes equations are solved using a second order Total Variation Diminishing (TVD) scheme for the inviscid flux.

2. GOVERNING EQUATIONS

We consider a system of conservation laws written in integral form for a moving and deforming control volume:

$$\frac{d}{dt}\int_\Omega U dV \; + \; \int_{\partial\Omega} \left(\vec{F} - U\dot{\vec{r}}\right)\cdot \vec{n}\; ds \; = \; 0 \tag{1}$$

Here U is the conserved variable, $\vec{F}$ the flux vector and $\vec{n}$ the outwards pointing normal to the surface $\partial\Omega$ enclosing an arbitrary volume Ω, and $\dot{\vec{r}}$ the velocity of the point $\vec{r}$ on $\partial\Omega$. We assume that the flux vector can be split into one inviscid and one viscous part, where the Jacobian of inviscid flux with respect to U has real eigenvalues and a complete set of eigenvectors, and that the viscous flux depends linearly on ∇U as for a Newtonian fluid.

In addition we make the assumptions that the fluid is a perfect gas with $\gamma = 1.4$, that the viscosity is constant, that Stokes hypothesis is valid, and that thermal conductivity is proportional to the viscosity.

3. SPACE DISCRETIZATION

The convective part of fluxes are discretized with a TVD scheme based on Roe's scheme with second order Monotone Upwind Schemes for Conservation Laws (MUSCL) extrapolation [4]. Considering a system of non-linear hyperbolic equations in one dimension, this scheme is briefly reviewed here. The extension to two or three dimensions is straight forward by applying the one-dimensional scheme independently in each coordinate direction.

For ease of notation we write $F(U) = (\vec{F}(U) - U\dot{\vec{r}}) \cdot \vec{n}$ and let $F_{i+1/2}$ denote the flux trough the surface $S_{i+1/2}$. By Roe's scheme, the numerical flux is then given by

$$F_{i+\frac{1}{2}} = \frac{1}{2}\left(F(U^-_{i+\frac{1}{2}}) + F(U^+_{i+\frac{1}{2}})\right) - \frac{1}{2}(A^+_{i+\frac{1}{2}} - A^-_{i+\frac{1}{2}})(U^+_{i+\frac{1}{2}} - U^-_{i+\frac{1}{2}}) \tag{2}$$

Here $U^-_{i+1/2}$ and $U^+_{i+1/2}$ are extrapolated values of the conserved variable at the left and right side of the surface, $\dot{\vec{r}}_{i+\frac{1}{2}}$ the velocity of the surface, and $A^\pm_{i+1/2}$ are given by

$$A^\pm_{i+\frac{1}{2}} = R_{i+\frac{1}{2}}\Lambda^\pm_{i+\frac{1}{2}}R^{-1}_{i+\frac{1}{2}} \tag{3}$$

where $R_{i+1/2}$ is the right eigenvector matrix of $A_{i+1/2}$ which is the Jacobian of $F(U)$ with respect to U taken at $U_{i+1/2}$. The state $U_{i+1/2}$ is given by Roe's approximate Riemann solver. The diagonal matrices $\Lambda^\pm_{i+1/2}$ are the split eigenvalue matrices corresponding to $R_{i+1/2}$. To avoid entropy violating solutions, the eigenvalues are split according to:

$$\lambda^\pm_l = \frac{1}{2}\left(\lambda_l \pm \sqrt{\lambda_l^2 + \varepsilon^2}\right) \tag{4}$$

where ε is a small parameter.

The first order version of Roe's scheme is obtained by simply setting $U^-_{i+1/2} = U_i$ and $U^+_{i+1/2} = U_{i+1}$. Schemes of higher order are obtained by defining $U^-_{i+1/2}$ and $U^+_{i+1/2}$ by higher order extrapolation. The scheme used here is based on a second order "minmod" limited extrapolation of the characteristic variables resulting in a scheme that is fully upwind.

For the resulting time integration to be conservative for a moving mesh, we must have conservation of volume [5]. Here, this is fulfilled by using the following expression for the mesh velocity $\dot{\vec{r}}_{i+\frac{1}{2}}$ at the surface $S_{i+\frac{1}{2}}$:

$$\dot{\vec{r}}_{i+\frac{1}{2}} \cdot \vec{S}_{i+\frac{1}{2}}\Delta t = \Delta V^n_{i+\frac{1}{2}} \tag{5}$$

where $\vec{S}_{i+1/2}$ is the surface vector and $\Delta V^n_{i+\frac{1}{2}}$ is the volume of the hexahedron defined by having the surface $S_{i+\frac{1}{2}}$ at the two time levels n and $n+1$ as opposite sides.

The viscous fluxes are calculated using central differencing.

4. TIME INTEGRATION

Implicit and second order accurate time stepping is achieved by a three point A-stable linear multistep method [6]:

$$\frac{3}{2\Delta t}U_i^{n+1}V_i^{n+1} - \frac{2}{\Delta t}U_i^nV_i^n + \frac{1}{2\Delta t}U_i^{n-1}V_i^{n-1} = R^{n+1}\left(U_i^{n+1}V_i^{n+1}\right) \tag{6}$$

Here $R\left(U_i^{n+1}V_i^{n+1}\right)$ is the sum of the flux contributions, V_i the volume of the grid cell i, Δt the time step, and the superscript n refers to the time level.

Equation 6 is solved by an approximate Newton iteration. Using l as the iteration index, it is customary to introduce a modified residual

$$R^*(U) = \frac{3}{2\Delta t}U_i^{n+1}V_i^{n+1} - \frac{2}{\Delta t}U_i^nV_i^n + \frac{1}{2\Delta t}U_i^{n-1}V_i^{n-1} \tag{7}$$

that is to be driven to zero at each time step by the iterative procedure:

$$\left(\partial R/\partial U + \frac{3}{2\Delta t}V_i^{n+1}\right)\Delta U + R^*(U^l) = 0 \tag{8}$$

updating at each iteration $U^{l+1} = U^l + \Delta U$. The Newton procedure is approximate because the the flux vectors are approximately linearized based on the first order version of Roe's scheme.

The resulting linear system that has to be solved for each iteration of the Newton procedure is septa-diagonal in each block. Ignoring the block interface conditions, this system is solved concurrently in each block using a either a line Jacobi procedure [2] or Lower Upper Symmetric Gauss-Seidel (LU-SGS). A coarse grid correction scheme [3] is used to compensate for ignoring the block interface conditions by adding global influence to the solution. The coarse mesh is obtained by dividing the fine mesh into a suitable number of cells by removing grid lines. The size of the coarse grid system is chosen as the maximum that can be solved efficiently with a parallel solver.

Using the subscript h to denote the fine mesh and H for the coarse mesh, the method is described in the following. In matrix form, we can write the linear system of equations that we wish to solve at each Newton iteration as

$$A_h\Delta U_h = R_h \tag{9}$$

However, by ignoring the coupling between blocks, we actually solve a different system that we symbolically can write:

$$\tilde{A}_h\widetilde{\Delta U}_h = R_h \tag{10}$$

Defining the error obtained by not solving the correct system as $\varepsilon_h = \Delta U_h - \widetilde{\Delta U}_h$ we have

$$A_h\varepsilon_h = R_h - A_h\widetilde{\Delta U}_h \tag{11}$$

In a classic multigrid fashion we now formulate a coarse grid representation of Equation 11 based on a restriction operator I_h^H. Thus we define

$$A_H = I_h^H A_h I_H^h \tag{12}$$

$$R_H = I_h^H\left(R_h - A_h\widetilde{\Delta U}_h\right) \tag{13}$$

and solve for the correction to $\widetilde{\Delta U}_h$:

$$A_H \Delta U_H = R_H \tag{14}$$

The solution to Equation 14 is transformed to the fine grid by means of the prolongation operator I_H^h, and finally the fine grid solution is updated by:

$$\widetilde{\widetilde{\Delta U}}_h = \widetilde{\Delta U}_h + I_H^h \Delta U_H \tag{15}$$

As restriction operator we use summation and as prolongation operator injection [7].

To solve the coarse grid system, any suitable parallel sparse matrix solver can be applied. We have used a Jacobi type iterative solver that at each iteration inverts only the diagonal coefficients in the coarse grid system.

5. VORTEX SHEDDING FROM A CIRCULAR CYLINDER

The first test case is the flow around a circular cylinder at vortex shedding from a circular cylinder at Reynolds numbers from 100 to 200 for which the flow is unsteady, laminar, and at least in the lower range of Reynolds numbers, truly two dimensional. The Mach number in these calculations was set to 0.3 to simulate nearly incompressible flow. A mesh consisting of 128 × 128 cells divided into 32 blocks was used. The reduced time step was 0.1 which corresponds to a CFL number of about 3000 and about 50 time steps per shedding cycle.

Figure 1. Calculated instantaneous entropy field behind a circular cylinder.

The results, in terms of the non-dimensionalized frequency, or Strouhal number, of the lift is shown in Figure 2 and compared to experimental data [8] and calculations by other

authors [9]. A typical convergence history for the approximate Newton iterations using the CGCS is shown in Figure 3 and compared to the convergence history using a block Jacobi approach. The CPU time refers to 32 nodes on an Intel Paragon using LU-SGS to invert the linear system of equations. We can clearly see the improvement obtained by using CGCS, and the need for a globally coupled solution procedure. Especially after about a two decades reduction of the residual the difference is noticeable, which is almost certainly due to inferior damping of error modes with long wavelengths with the basic block Jacobi procedure without the CGCS. Based on results from steady state computations, we expect the difference between the two methods to be larger as the number of blocks is increased.

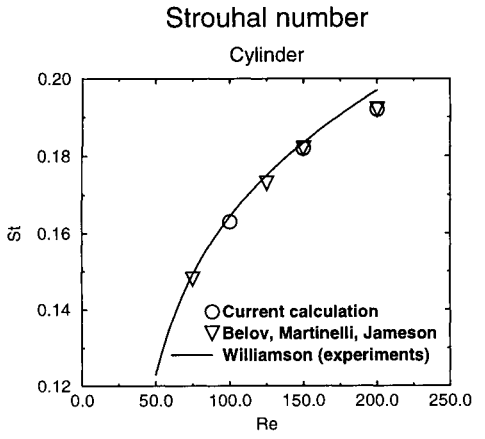


Figure 2. Calculated Strouhal number.

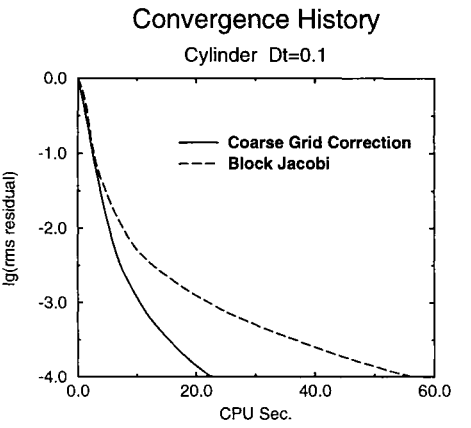


Figure 3. Typical convergence history for the Newton iterations.

6. FLOW OVER SUSPENSION BRIDGE DECKS

As an example of an industrial relevant test case, we have included some some simulations of the flow over a suspension bridge deck. In a typical analysis of the aerodynamics of a large span bridge, the first step is to calculate the lift, drag, and torque coefficients of a cross section of the bridge span for wind at different angles of attack. In Figure 4 we show the time average of the calculated the lift, drag, and torque coefficients for the cross section of the Askøy suspension bridge. These calculations were based on the Navier-Stokes equations without any form for turbulence modeling. In spite of this obvious deficiency, the results agree reasonably well with experiments [10].

The results were obtained using a reduced time step of 0.02 based on the width of the cross section and the free stream velocity. The calculations were carried out for a Mach number of 0.3, resulting in a maximum CFL number of approximately 1000.

Of increased complexity is the calculation of the dynamic behavior of the bridge for a given wind condition, requiring the CFD code to handle moving boundaries and thereby

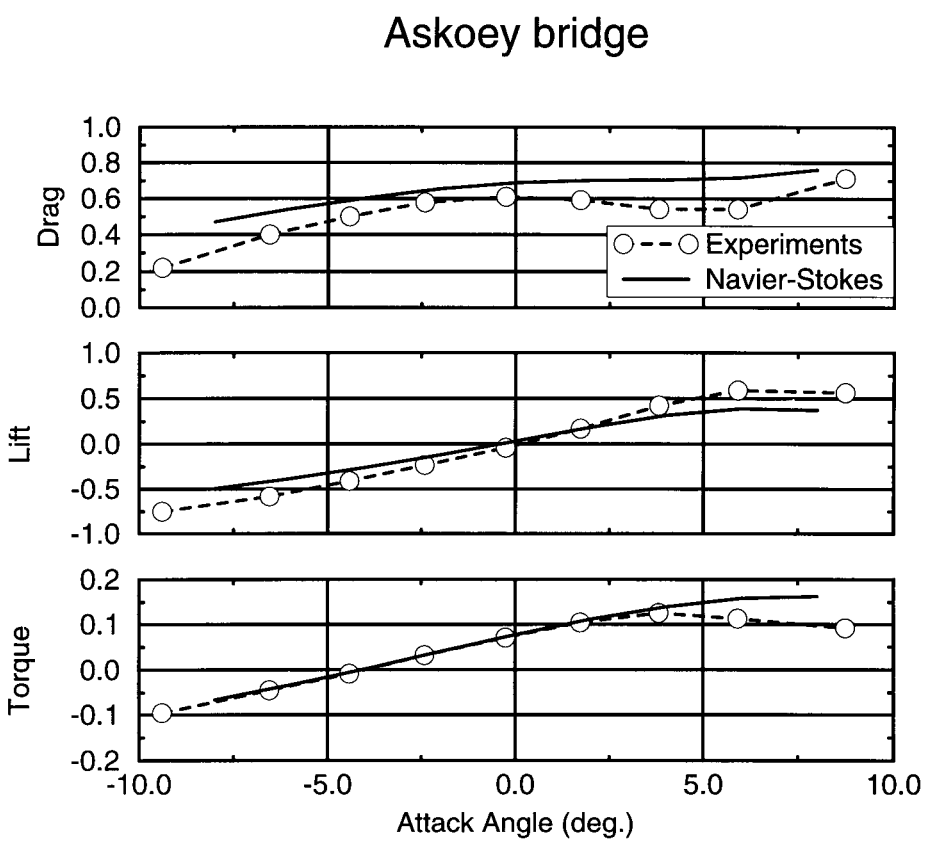


Figure 4. Calculated forces on the cross section of the Askøy suspension bridge.

also a moving computational mesh. In addition, the CFD code must be coupled to a Computational Structure Dynamics (CSD) code for predicting the dynamic response of the structure. An example of such an aeroelastic calculation is shown in Figure 5, where we have calculated the pivoting motion of a cross section suspended on an axis with a given torsional elasticity. In this two-dimensional case, the CSD code was replaced by a simple second order ordinary differential equation.

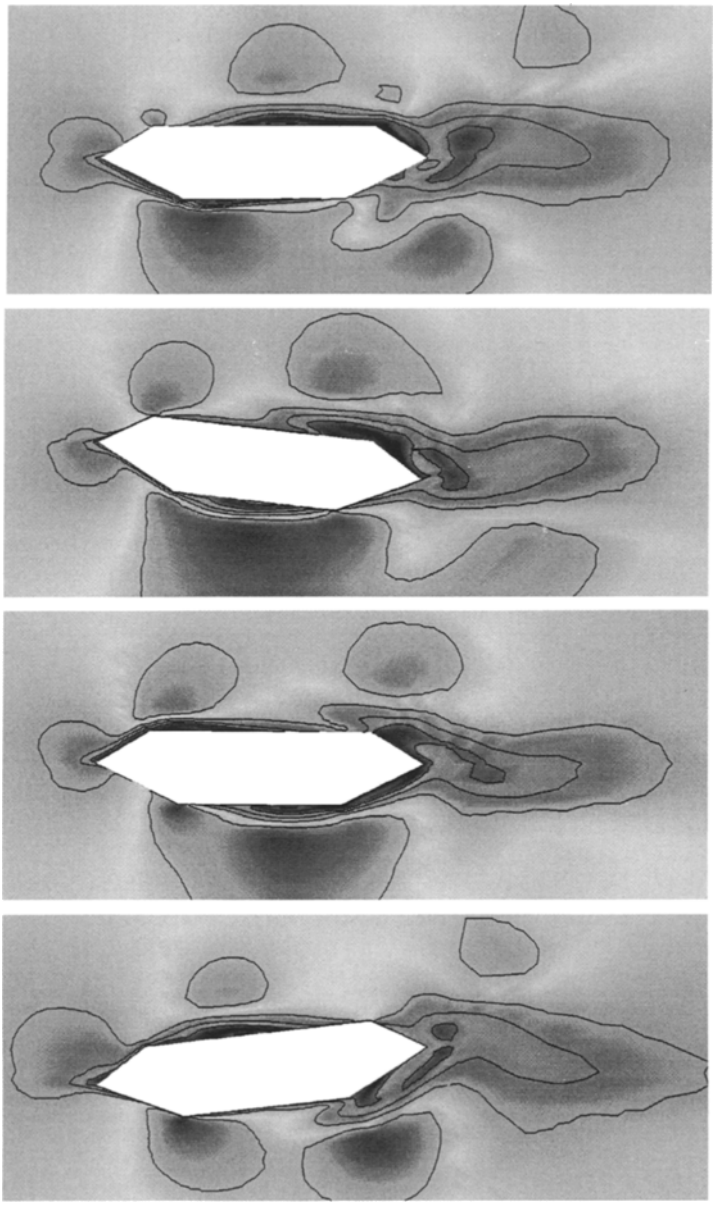


Figure 5. Velocity field around bridge cross section oscillating as a result of vortex shed ding.

7. CONCLUSION

We have seen that the Coarse Grid Correction scheme for implicit multiblock computations can be extended to time accurate Navier-Stokes calculations. Using 32 blocks, the CPU time required for a given level of accuracy is reduced by at least a factor of two compared to using explicit coupling between the blocks, emphasising the need for a globally coupled solution procedure.

The use of an A-stable implicit method for time dependent computations allows the time step to chosen according to the physics of the flow, instead of on the basis numerical stability. Thus CFL numbers higher than 1000 have been used in this work.

As an example of realistic flow problem, we have calculated the fluid-structure-interaction of a pitching cross-section of a suspension bridge.

Future work will include preconditioning to increase the efficiency for low Mach numbers, as well as Large Eddy Simulations to more accurately predict time dependent flows of industrial relevance.

REFERENCES

1. C.B. Jenssen. A parallel block-Jacobi Euler and Navier-Stokes solver. In *Parallel Computational Fluid Dynamics: New Trends and Advances, Proceedings of the Parallel CFD'93 Conference*, pages 307–314. Elsevier, Amsterdam, The Netherlands, 1995.
2. C.B. Jenssen. Implicit multi block Euler and Navier-Stokes calculations. *AIAA Journal*, 32(9):1808–1814, 1994. First published as AIAA Paper 94-0521.
3. C.B. Jenssen and P.Å. Weinerfelt. A coarse grid correction scheme for implicit multi block Euler calculations. Presented as AIAA Paper 95-0225, Jan. 1995. Accepted for publication in the AIAA Journal.
4. S.R. Chakravarthy. High resolution upwind formulations for the Navier-Stokes equations. In *Proceedings from the VKI Lecture Series 1988-05.* Von Karman Institute of Fluid Dynamics, Brussels, Belgium, March 1989.
5. J. Batina. Unsteady Euler algorithm with unstructured dynamic mesh for complex-aircraft aerodynamic analysis. *AIAA Journal*, 29(3):327–333, 1991.
6. J.D. Lambert. *Computational Methods in Ordinary Differential Equations.* John Wiley and Sons, New York, 1973.
7. P. Wesseling. *An Introduction to Multigrid Methods.* John Wiley and Sons, New York, New York, 1992.
8. C.H.K. Williamson. Defining a universal and continious Strouhal-Reynolds number relationship for the laminar vortex shedding of a circular cylinder. *Phys. Fluids*, 31:2742–2744, 1988.
9. A. Belov, L. Martinelli, and A. Jameson. A new implicit algorithm with multigrid for unsteady incompressible flow calculations. AIAA Paper 95-0049, Jan. 1995.
10. E. Hjort-Hansen. Askøy bridge wind tunnel tests of time-average wind loads for box-girder bridge decks. Technical Report STF71 A87037, SINTEF, 7034 Trondheim, Norway, 1987.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel implementation of Newton's method for 3-D Navier-Stokes equations

X. Xu and B. E. Richards

Department of Aerospace Engineering, University of Glasgow, Glasgow G12 8QQ, U.K.

Newton's method, an extreme candidate against the explicit method for solving discretized NS equations, is investigated on parallel computing environments. For a 3-D flow field, with a structured mesh, domain decomposition technique is used in two indices. Cell ordering is in the global field, and a Jacobian, which has 25-point block diagonal stencil, from Newton's method is also corresponding to the global field without any simplification. Generation of the preconditioner is imposed on an approximated Jacobian, which increases the ability to parallelise, and is based on the structure of the matrix. The technique that arranges the matrix elements, which keeps the most important information, to the position near the diagonal of Jacobian, is also used to enhance the efficiency of a block ILU preconditioner. Generation and storage of non-zero elements of Jacobian are fully parallel implemented. Cell centred finite volume method and Osher upwind scheme with third order MUSCL interpolation are used. The test physical flow is a supersonic flow around Ogive cylinder with angles of attack.

1. INTRODUCTION

The existence of flow interaction phenomena in hypersonic viscous flows requires the use of a high resolution scheme in the discretization of the Navier-Stokes (NS) equations for an accurate numerical simulation. However, high resolution scheme usually involve more complicated formulation and thus longer computation time per iteration as compared to the simpler central differencing scheme. Therefore, an important issue is to develop a fast convergent algorithm for the discretized NS equations.

For a steady state flow governed by locally conical NS (LCNS) or parabolized NS (PNS) equations, Newton's method has been proved as a robust and fast algorithm in solving the non-linear algebraic system from the discretized NS equations [1,2]. Parallel techniques were used to both accelerate calculation and store the Jacobian matrix. The most important issue was the development of a very efficient preconditioner, which made the algorithm not only simple for implementation but also very robust and fast convergent.

This paper communicate development of the Newton's method to the 3-D flow case, with the techniques developed for the LCNS and PNS solvers. The physical problem discussed is a steady state hypersonic viscous flow around Ogive cylinder with angle of attack.

2. DISCRETIZATIONS

Using a numerical method to solve a system of differential equations often includes the discretizations of the equations and the domain where the equations are defined.

2.1. Spatial discretization and domain decomposition

The flow field is discretized by a structured grid illustrated in figure 1. A domain decomposition technique is used in two directions J and K, which are corresponding to circumferential and axial directions of the cylinder respectively. The cell ordering is according to (i,j,k), where i is the inner index and k is the outer index, in the global grid.

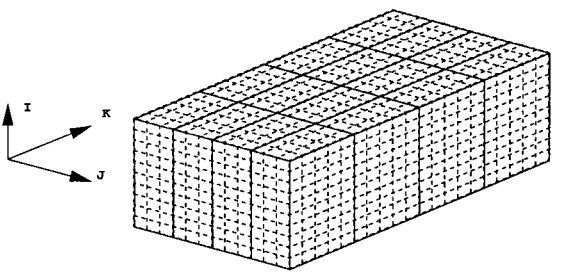


Fig. 1 3-D grid with domain decomposition

IN, JNTO, and KNTO are the all number of global grid points in i, j, and k directions respectively. Therefore, the global cells in the above directions, in which the unknown are set, are from 2 to IN, 2 to JNTO, and 2 to KNTO. In each subdomain, IN, JN, KN denote the number of grid points in i, j, and k directions respectively, and the cells, in which the unknown are set, are from 2 to IN, 2 to JN, and 2 to KN.

2.2. Discretized NS equations

The cell centred finite volume and finite difference formulation are used for the convective and diffusive interface flux calculations respectively. The Osher flux difference splitting scheme is used with MUSCL high order interpolation.

On a cell (i,j,k) the discretized equation is

$$R_{cell(i,j,k)}(V) = 0 \tag{1}$$

where R is the residual and V is physical variables.

The extent of V in the calculation of R in cell (i,j,k) is illustrated in figure 2, which includes 25 cells. The update of R in cell (i,j,k) can be called with local properties.

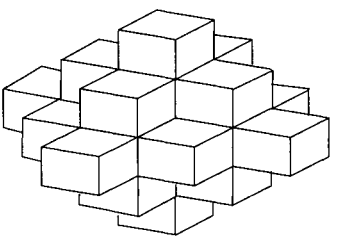


Fig. 2 Extent of variables V

Consolidating all the discretized NS equations in every cell we have a non-linear algebraic equation as follows:

$$R(V) = 0 \tag{2}$$

3. NEWTON'S FORMULATION

For Eq. 2, the general Newton's method is

$$(\partial R(V)/\partial V)^k \Delta^k V = - R(V^k) \tag{3}$$
$$V^{k+1} = V^k + \Delta^k V$$

For Eq. 1, we have the Newton's formulation in each cell (i,j,k) as follows:

$$(\partial R_{cell(i,j,k)}(V)/\partial V)^k \Delta^k V = - R_{cell(i,j,k)}(V^k) \tag{4}$$

For any $V_{cell(l,m,n)}$, $\partial R_{cell(i,j,k)}(V) \setminus \partial V_{cell(l,m,n)}$ is a 5×5 submatrix, and will be zero for the cell (l,m,n) out of the 25 cells in the Figure 2. Therefore the Jacobian of Eq. 2 is a 25-point block diagonal matrix, and each block here is a 5×5 submatrix. A numerical Jacobian can be generated by using a difference quotient instead of the above differential formulation.

The Jacobian elements can be further grouped and written as a 5 block diagonal matrix as in figure 3. In the figure each element from left to right in a row corresponds to the residual

vector within constant k section and the variables vector has perturbations within constant k-2, k-1, k, k+1, or k+2 section respectively.

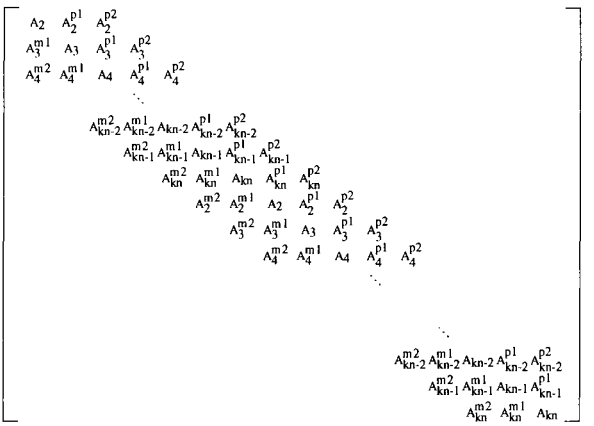


Fig. 3 Jacobian of the Newton's method

We denote the linear system in Eq. 5 by

$$A\,x = b \tag{5}$$

4. LINEAR SOLVERS

4.1. Linear system

For the equation 5, a preconditioning technique can be used. Let L^{-1} and U^{-1} be left and right preconditioners, the equation will be as follows:

$$\begin{aligned} L^{-1} A U^{-1} y &= L^{-1} b \\ U^{-1} x &= y \end{aligned} \tag{6}$$

Therefore the procedure for solving Eq. 5 is divided into three steps.

Step 1: Set new right hand side vector

$$B = L^{-1} b$$

Step 2: Obtain y

Step 3: Obtain solution of Eq. 5 by

$$U\,x = y$$

Step 1 and 3 correspond to solving the lower and upper linear equations respectively. For solving y in step 2 we use the GMRES linear solver in which the new matrix, $C = L^{-1} A U^{-1}$, and vector manipulation can be implemented by (1) solving an upper linear equation, (2) matrix vector manipulation and (3) solving a lower linear equation.

4.2. Preconditioner

The ILU factorization of the Jacobian is a usual and efficient method for generating the preconditioner. However, some important things needing to be considered in the generation of preconditioner are (1) keeping the most important information of the Jacobian in the lower and upper matrics, (2) avoiding fill-in, to simplify the procedure and save storage, (3) making the procedure of generating the preconditioner and solving the linear system, more parallelizable, and (4) using a structure of the Jacobian to save computation time.

From the tests for LCNS and PNS equations, we know that these block elements of the 25-points block diagonal Jacobian, that correspond to the cells along the direction from solid boundary to the far field (i.e. i direction in grid), involve the most important information of the Jacobian. Therefore, we should arrange the i direction as the inner index in the Jacobian elements ordering. In this case, these block elements are concentrated to positions near the diagonal of the Jacobian, and ILU factorization can keep them better. By using this technique, the preconditioner is robust even without any fill-in.

Since the Jacobian has a structure, block ILU is used according to this structure. The block ILU can be generated from the Jacobian itself directly, but more sequential bottle-necks are involved. One way to tackle this problem is that block ILU will be generated from an approximated Jacobian as figure 4.

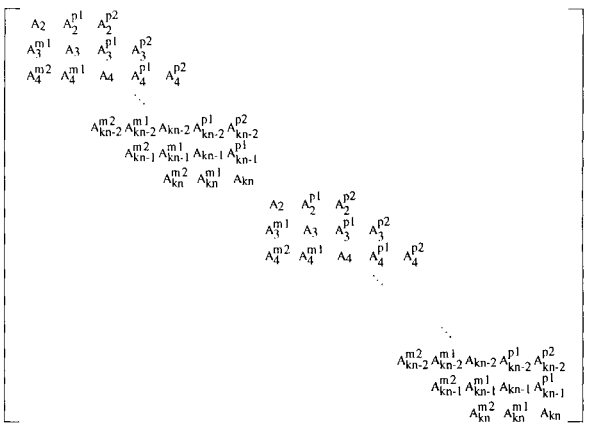


Fig. 4 Approximation of the Jacobian

Therefore, the lower and upper matrics will have the form as in figures 5 and 6. From the numerical tests of this paper, we can see that the method has very good efficiency and ability of parallelise . Another more parallelisable method is that the block ILU will be generated from an approximated Jacobian just with the diagonal block in figure 3, however practical calculation shows that it will lose efficiency.

The linear solver used is GMRES solver [3]. The CGS [4] solver has also been tested, but no convergence result was obtained for this linear solver.

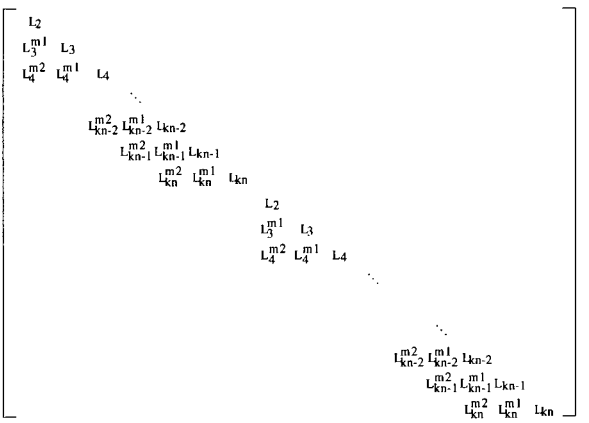


Fig. 5 Lower factorization of the approximate Jacobian

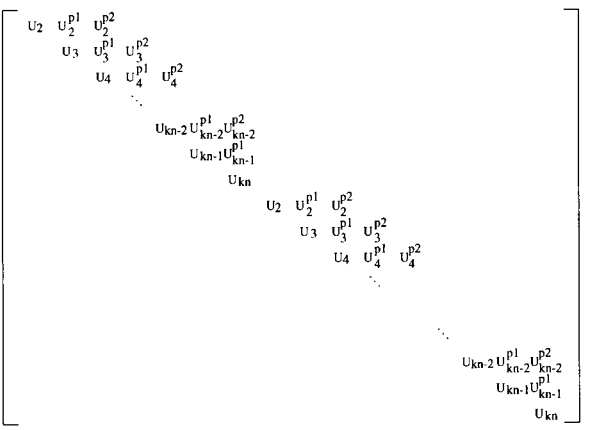


Fig. 6 Upper factorization of the approximate Jacobian

5. NUMERICAL RESULTS

The foregoing numerical tests have been carried out to solve the 3-D NS equation for compressible flow. The case is the flow around an Ogive cylinder at Mach 2.5 and an angle of attack of 14°. The test case produces a flow which has a large separated flow region with an embedded shock wave in the leeward side of the object and strong gradients in the thin boundary layer on the windward side, more details of the physical phenomena involved in these flows can be found in [5,6]. The parallel computer used is CRAY T-3D of the Edinburgh Parallel Computing Centre.

Fig. 7 shows the convergence results for the global grid size 33×33×17, where IN, JNTO, and KNTO are 33, 33, and 17. The grid in the j direction is decomposed into 8 parts, and in the k direction are decomposed into 1, 2, or 4 parts respectively. Fig. 8 shows the parallel efficiency achieved for the decompositions in the k direction.

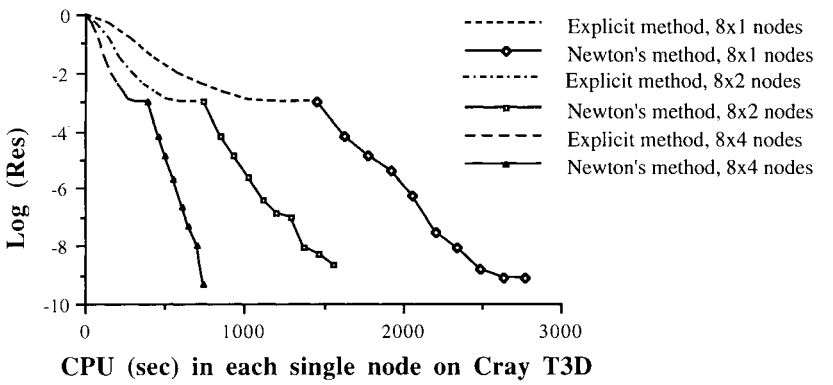


Fig. 7 Convergence histories for grid 33×33×17 case

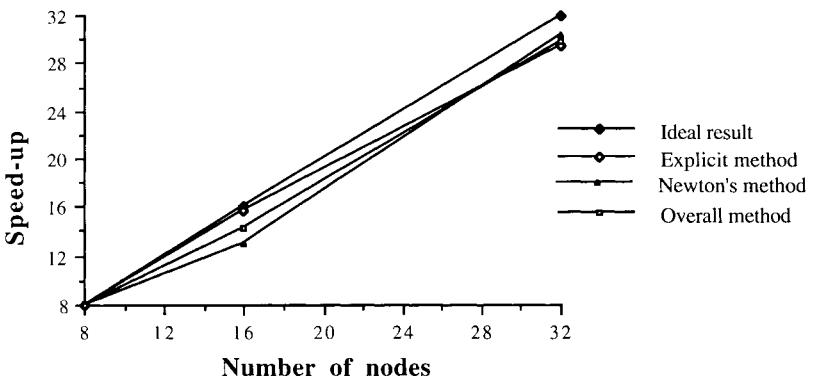


Fig. 8 Parallel efficiency

Fig. 9 and 10 show the convergence results for 33×65×17 and 33×33×33 grid cases.

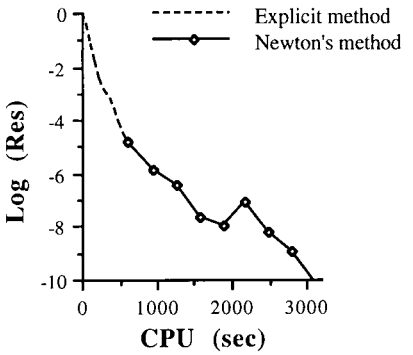


Fig. 9 33×65×17 grid, 16×4 processors

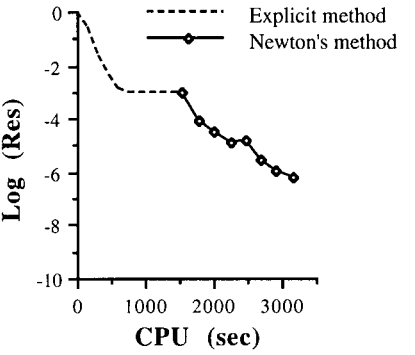


Fig. 10 33×33×33 grid, 8×4 processors

6. CONCLUSION

Newton's method is still robust for solving 3-D NS equations with the grid sizes used in this paper, and high parallel efficiency has been achieved. The preconditioner used is robust, inexpensive, and very parallelise well. However, it is difficult to solve the linear system when the rank of the global Jacobian is increased. Quadratic convergence was not achieved by the Newton's method since it is very expensive for the linear solver to converge sufficiently well.

REFERENCE

1. X. Xu, N. Qin and B.E. Richards, Parallelising Explicit and Fully Implicit Navier-Stokes Solutions for Compressible Flows, Parallel Computational Fluid Dynamics '92, R.B. Pelz (ed.), Elsevier, (1993).
2. X. Xu and B.E. Richards, Numerical Advances in Newton Solvers for the Navier-Stokes Equations, Computational Fluid Dynamics '94, S. Wagner, J. Periaux, and E.H. Hirschel (eds.), Wiley, (1994).
3. Y. Saad and M.H. Schultz, SIAM J. Stat. Comp., Vol. 7, No. 3, 856, (1986).
4. P. Sonneweld, P. Wesseling, and P.M. de Zeeuw, Multigrid Methods for Integral and Differential Equations, D.J. Paddon and M. Holstein (eds.), Claredon Press, 117, (1985).
5. N. Qin, X. Xu and B.E. Richards, SFDN-α-GMRES and SQN-α-GMRES Methods for Fast High Resolution NS Simulations, ICFD Conference on Numerical Methods for Fluid Dynamics, K.W. Morton (ed.), Oxford University Press, (1992).
6. X.S. Jin, A Three Dimensional Parabolized Navier-Stokes Equations Solver with Turbulence Modelling, Glasgow University Aero Report 9315, (1993).

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Dynamic load balancing in a 2D parallel Delaunay mesh generator

N.A. Verhoeven*, N.P. Weatherill and K. Morgan

Department of Civil Engineering, University of Wales Swansea, Singleton Park, Swansea SA2 8PP, UK.

The rise of multi–processor computers has brought about a need for flexible parallel codes. A parallel mesh generator can make efficient use of such machines and deal with large problems. The parallelisation of the input data while using the original sequential mesh generator on different processors simultaneously is applied in a manager/worker structure. Sending a single task to a workstation can result in an imbalance. Dynamic load balancing is called upon to avoid this.

1. INTRODUCTION

Very fast mesh generators have always been at the heart of pre–processing for the finite element method. For the current generation of super computers the mesh generator is not considered a large consumer of processor time. There is however a need to solve problems where mesh sizes are extremely large and have to be generated several times over. Parallelisation can offer a solution for large grids.

A further incentive has been the arrival of low-cost parallel machines: parallel codes make more efficient use of multi–processor architectures. Other advantages are the access to large numbers of processors and large amounts of computer memory.

This paper presents an interim report on the parallelisation of a Delaunay–based mesh generator of Weatherill [1]. The code used is that for two–dimensional meshes. The parallelisation follows a manager/worker structure in which the workers execute the mesh generator subroutines. The input data for the mesh generator are the boundary nodes of the domain and their connectivity.

Hodgson et al. [2] use a coarsening of the boundary data to create an initial mesh. Recursive spectral bisection with weighted correction provides different sub-domains for parallel mesh generation. Topping et al. [3] apply a neural network and genetic algorithm. Both methods show good load-balancing and require minimal communication; they can be classified as graph-based.

The method presented in this paper is geometry-based: the geometrical data of the boundary is used to create artificial inter-zonal boundaries. Poor load balance is a possible disadvantage but can be corrected by a dynamic load-balancing scheme. Communication overhead is not considered.

Only distributed computing is discussed, though genuine parallel computers can be used as well.

*E–mail: N.Verhoeven@swansea.ac.uk
EPSRC UK funded research

2. MANAGER/WORKER MODEL

Figure 1 demonstrates the structure of the manager/worker model. The first block represents the reading of the input data. The input is subsequently divided into different workloads for the available workstations. Worker programs on the workstations do the actual mesh generation. A workstation which runs both the manager and a single worker code is referred to as the working manager. The results from the different workers are combined to give a single mesh (option 1). This model has been implemented by Weatherill and Gaither [5]. The computer memory of the working manager limits the size of the combined mesh.

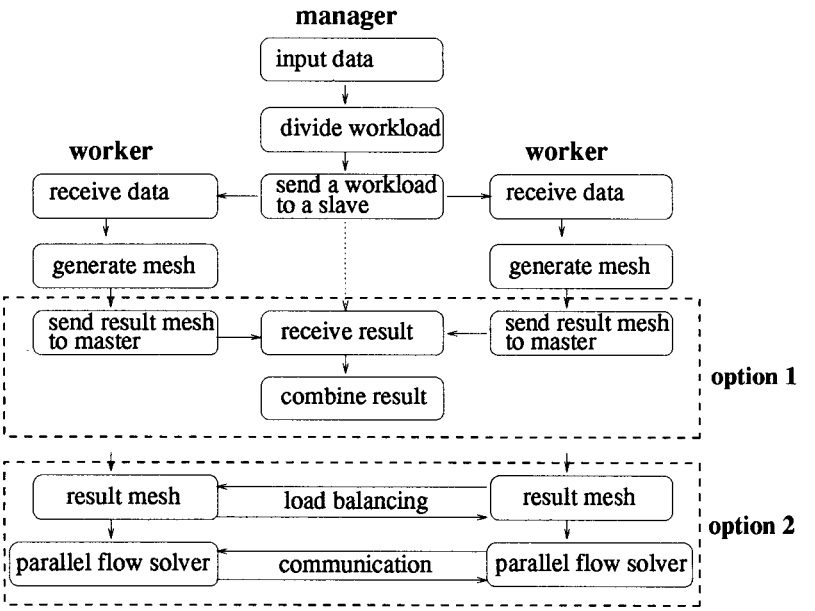


Figure 1. Structure of manager/worker code. Option 1: The sub-domain meshes are combined to a single mesh. Option 2: After load balancing a parallel flow solver is employed.

If the sub-domain meshes are processed separately the memory of the workstations can be used to the full extent (option 2). Load-balancing and minimal communication require a correction of the number of elements and the inter-zonal boundaries. This enables the effective use of a parallel flow solver. An example of post-processing load-balancing will be given in section 5.

3. DOMAIN BISECTION

To send data to a worker program the initial workload has to be divided. In the geometrical approach, the initial workload is the boundary data (see figure 2). A Delaunay mesh (see figure 3) - connecting all the boundary nodes - is used for domain

bisectioning. A greedy-like algorithm [4] and area criterion gives a number of equally-sized sub-domains. The boundaries of the sub-domains are the workloads for the worker programs. These must have node distributions related to their end-points and surroundings, as demonstrated in figure 4. The enlargement shows the node distribution on the inter-zonal boundaries. For the airfoil-to-airfoil boundary the surrounding has to be taken into account.

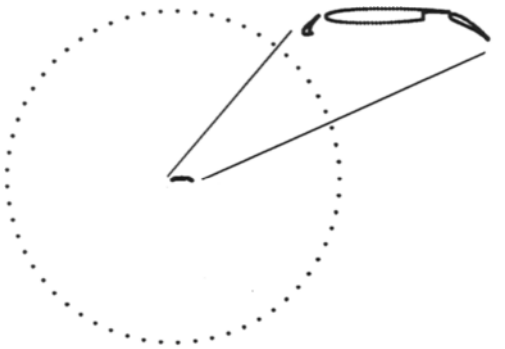


Figure 2. Boundary data of an airfoil

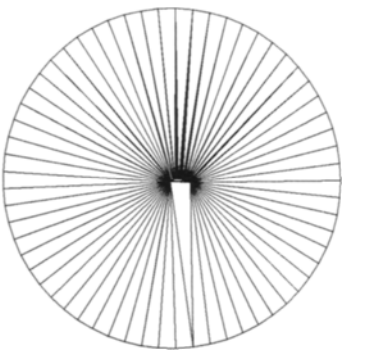


Figure 3. Initial mesh connecting the boundary nodes by using Delaunay

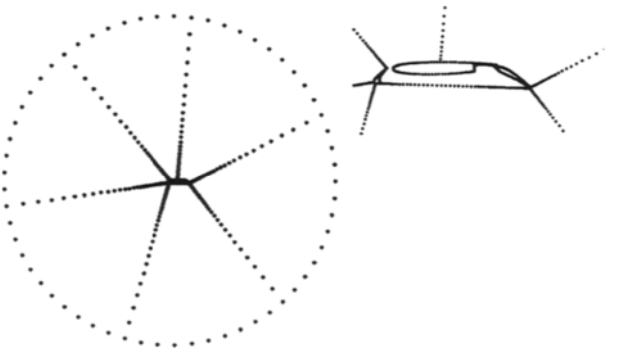


Figure 4. The introduction of artificial boundaries in domain bisectioning

4. DYNAMIC LOAD BALANCING

So far, the manager/worker code divides the domain into a number equal to that of the available processors. Figure 5 shows that sub-domains can have different workloads of variable complexity. Executing this problem on a workstation cluster of N identical machines means that the process is delayed by the workstation with the largest workload. This is demonstrated in figure 6 for a two workstation/sub-domain example: the dark bars represent active computing and the white the waiting time. Restructuring the code to execute the workloads in sequence on the same workstation makes better use of the

hardware, as demonstrated in figure 7. This method of dynamic loading, is possible because there is no communication between the worker programs, i.e. each sub-domain is generated independently.

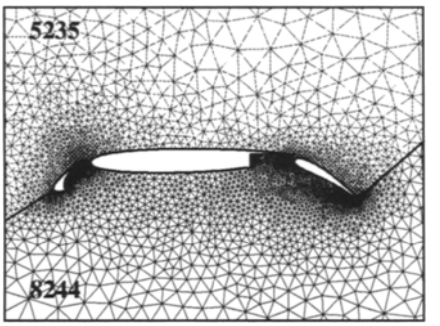


Figure 5. Two sub-domain mesh with a total of 13479 elements

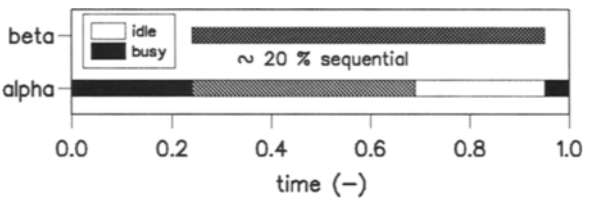


Figure 6. Execution and waiting times of a working manager (alpha) and worker (beta) workstation

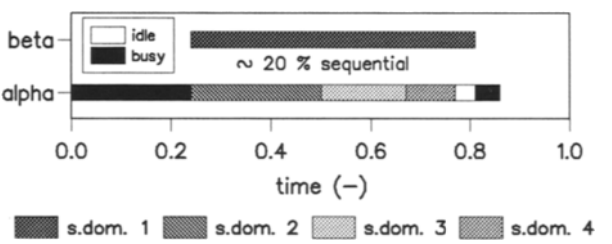


Figure 7. Execution and waiting times of a working manager (alpha) and worker (beta) workstation with dynamic loading of four sub-domains

The following listing, in pseudo–Fortran, shows how the manager code has to be structured to include dynamic loading.

```
program manager
initialisation( find number_of_processors)
split_input_in( number_of_workloads)
do k=1,number_of_processors
 send_a_workload_to_an_idle_processor
```

```
end do
do i=number_of_processors+1,number_of_workloads
 receive_a_finished_result_from_a_processor
 send_a_new_workload_to_this_processor
end do
do k=1,number_of_processors
 receive_a_finished_result_from_a_processor
 send_stop_signal_to_this_processor
end do
final_combination_of_result
end
```

The code first initiates a search for the number of workstations available and subsequently splits up the input data. Next, a do-loop sends the different workloads to the worker programs. The final do-loop receives the data from the worker programs. Static loading applies if 'number_of_processors' is 'number_of_workloads'. Dynamic loading takes place when larger numbers of workloads are involved, in which case the second do-loop is executed. Dynamic load-balancing comes into play if the sub-domain boundaries are sent to the worker programs in descending order of their estimated workloads. The number of boundary nodes provides a reasonable estimation. For a parallel virtual machine network the individual speed of workstations should be taken into account.

5. POST PROCESSING

The sub-domain meshes are recombined to a single large mesh by the working manager, though they can be left on the individual workstations if so required. The combination of sub-domains on a single workstation is shown in figures 8 and 9. Dynamic load balancing and mesh combination results in meshes with near equal element numbers on each workstation. The 'old' inter-zonal boundaries in the 'new' sub-domains are smoothed with a Gaussian correction scheme.

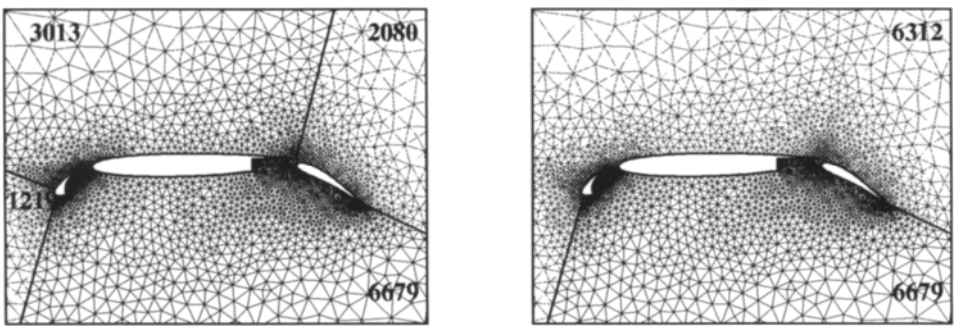


Figure 8. Four sub-domain meshes (left) recombined to two meshes (right) of near-equal element numbers (6312 and 6679)

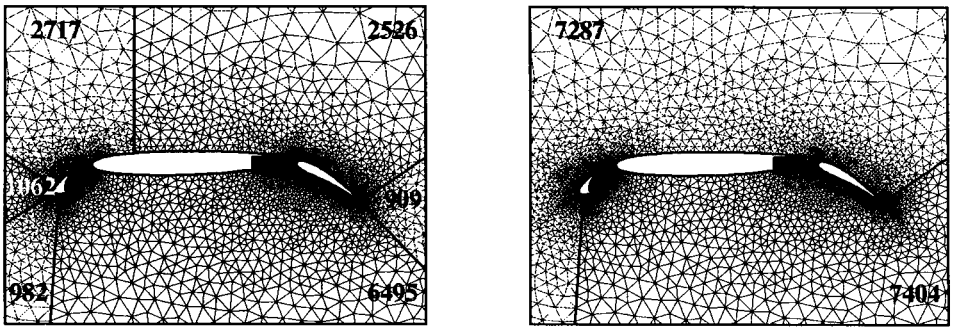


Figure 9. Six sub-domain meshes (left) recombined to two meshes (right) of near-equal element numbers (7287 and 7404)

6. MESSAGE PASSING

The message passing libraries used are PVM (Parallel Virtual Machine) [6] and MPI (Message Passing Interface) [7]. The parallel mesh generator code is structured in terms of a single program multiple data (SPMD). As a result, switching between PVM and MPI is relatively straightforward [8]. The basic subroutines in table 1 are sufficient for the parallel mesh generator code.

Table 1
message passing fortran subroutines

pvm3.3	mpich1.0	
pvmfmytid	MPI_Init	Start a message passing session
pvmfexit	MPI_Finalize	Stop the message passing session
pvmfspawn	...	Spawn child session
pvmfparent	MPI_Comm_rank	Determine the process identifier
pvmfpsend	MPI_Send	Send a message
pvmfprecv	MPI_Recv	Receive a message

7. SPEEDUP AND EFFICIENCY

Speedup and efficiency measurements are carried out on a homogeneous network of up to a maximum 5 workstations. In general, the speedup is defined as the ratio of the times in which a sequential code and its parallel equivalent finish computing. Here, speedup is defined as the time used by the parallel code to mesh an M sub-domain problem on a single workstation, divided by the time required on N workstations. This definition takes into

account the different numbers of elements generated for domains with different sets of sub-domains (see [9]). Figure 10 shows the speedup for different numbers of workstations. The dotted lines represent the maximum achievable speedup if 30% of the code is sequential.

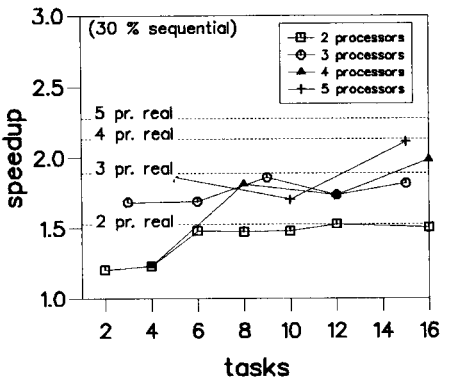


Figure 10. Speedup of a two, three, four and five workstation parallel code with 30% sequential part

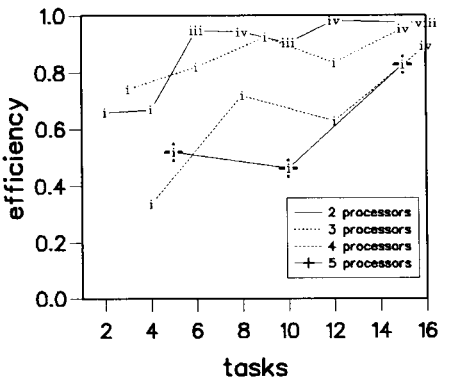


Figure 11. Efficiency of the parallel part of the code a function of the number of tasks

The main idea behind dynamic load balancing is to improve the use of the workstations. Efficiency is therefore defined as the total time that N workstations are active in parallel, divided by the product of N and the parallel time of the slowest workstation. Figure 11 shows the efficiency as a function of the number of tasks (i.e. sub-domains). The Roman numerals at the measurement points represent the number of tasks executed on the slowest workstation.

8. DISCUSSION

Dynamic load balancing can improve the efficiency of workstation use. For an N workstation configuration it is recommended to generate i times N tasks (i is an integer). This avoids any imbalance due to tasks representing similar or near similar workloads. The method also gives the opportunity to generate meshes larger than the size of computer memory. The sequential part of the code is still considerable and further research is recommended.

REFERENCES

1. N. P. Weatherill. A method for generating irregular computational grids in multiply connected planar domains. *International Journal for Numerical Methods in Fluids*, 8:181-197, 1988.
2. D. Hodgson, P. Jimack, P. Selwood, and M. Berzins. Scalable Parallel Generation of Partitioned, Unstructured Meshes In *Parallel CFD'95 Conference Abstracts*, Session 19-1, Caltech, Pasadena, USA.

3. B. H. V. Topping, A. I. Kahn and J. K Wilson Parallel Dynamic Relaxation and Domain Decomposition. In *Advances in Parallel and Vector Processing for Structural Mechanics*, 215–232, B. H. V. Topping, and M. Papadrakais (Editors), 1994
4. C. Farhat. A Simple and Efficient Automatic FEM Domain Decomposer. *Computer and Structures*, Vol. 28, No. 2, 579–602, 1988.
5. N. P. Weatherill and A. J. Gaither. Paper presented at Super Computing '94.
6. Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek and Viady Sunderam. PVM3 User's Guide and Reference Manual. Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, May 1994.
7. William Gropp, Edwing Lusk and Anthony Skjellum. Using MPI: Portable Parallel Programming with the Message Passing Interface. The MIT Press. Cambridge, Massachusetts. London, England. 1994.
8. N. A. Verhoeven, J. Jones, N. P. Weatherill, and K. Morgan. PVM and MPI applied to a master/slave parallel mesh generator. In *PPECC Workshop '95 Distributed vs Parallel?*, 99–105 Editor: Brian Henderson. Parallel & Distributed Systems Group, DRAL, Chilton, UK. 1995
9. N. A. Verhoeven, N. P. Weatherill, and K. Morgan. Interim stage in the development of a parallel mesh generator. In *3rd ACME Conference Computational Mechanics in the UK*, 51–54. UCINA, Oxford University Computing Laboratory, January 1995.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel Visualisation of Unstructured Grids

P. I. Crumpton[a] * and R. Haimes[b]

[a]Oxford University Computing Laboratory, Wolfson Building, Oxford, OX1 3QD

[b]Department of Aeronautics and Astronautics,
Massachusetts Institute of Technology, Cambridge, MA 02139

This paper demonstrates the visualisation package **pV3**, which is a significant aide to the development and debugging of parallel code. By using a straightforward FORTRAN interface, a user can receive invaluable visual information concerning a calculation on an unstructured 3D grid which is distributed over a number of processors.

To demonstrate this visualisation facility the OPlus library was used to parallelise some complex algorithms which use unstructured grids, such as multigrid.

INTRODUCTION

Over the last decade much time and effort has been invested in parallelising state-of-the-art CFD algorithms. These proceedings demonstrate the fruits of this effort which include the successful parallel implementation of: 3D Euler and Navier-Stokes Multiblock codes, 3D unstructured multigrid methods, adaptive grid methods with dynamic load balancing and numerous other examples. Many of the underlying "parallel" ideas in these codes are straightforward; splitting the grid into several parts, then distributing the parts onto different processors. However, the implementation of these ideas is known to be "difficult", especially when the CFD application programmer is coding with low level message passing libraries such as MPI and PVM. If the application programmer is responsible for the parallel visualisation, parallel algorithm implementation as well as algorithm development, then these machines are not an attractive prospect.

One alternative is to develop a high level parallel language where the parallelism can be easily extracted by an efficient compiler. This is a long term goal that has not yet been realised. The approach taken here is to develop a library which enables certain tasks to be performed with little parallel input from the user. That is **pV3** [6] for visualisation of data that is distributed across several processors, and OPlus [1] which enables the straightforward parallelisation of a class of unstructured grid algorithms.

It is not intended that this paper be in any way a user guide for **pV3** or OPlus, and it is not possible to show how easy either are to use. Further details of OPlus and **pV3** are available in these proceedings, [7] [4]. Here, a brief summary of the objectives and capabilities are outlined, followed by a number of examples of where these tools where used for debugging and production calculations.

*Supported by Rolls-Royce plc, DTI(uk)

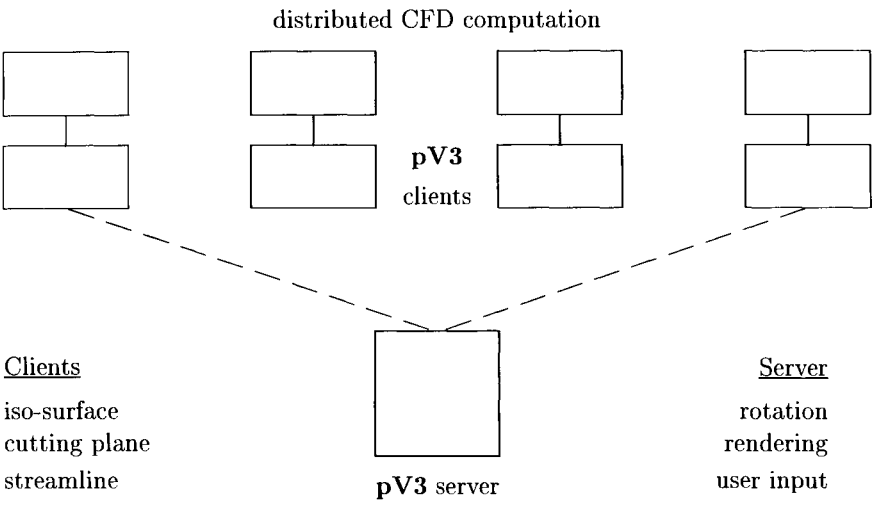


Figure 1. The **pV3** client-server model

1. THE pV3 LIBRARY

Although it may appear clear that interactively visualising the computation, while it is running on the MIMD machine is an attractive prospect, there is little evidence of "realistic" problems being treated this way in the literature.

The **pV3** software determines surface information from each process running on the parallel machine, and communicates this to a graphics workstation which renders and displays the image. The first important point is the graphics workstation does not need to hold all the grid information, consequently the memory burden for visualisation is on the distributed memory machine; which is easily met.

Another key feature is the ability to interrupt the parallel execution, visualise the current flow data, then stop the visualisation, while the parallel execution continues. Thus **pV3** gives the use the ability to "see" any data any time within the execution of the program.

1.1. Distributed Visualisation

The **pV3** visualisation system uses a client/server model, see Fig. 1. The clients are executing the application programmer's code on some parallel machine. The server is a graphics workstation, where the rendered image will be displayed. The following simple flow table outlines how the client and server interact.

1. The server takes interactive keyboard input describing the attributes of the surfaces required by the user and sends requests to the clients.

2. Each client receives this and collects all the 2D surface information within the current partition that is needed for the required display on the server, and then

sends it to the server. This may involve calculation of an iso-surface or cutting plane within the client's partition.

3. The server receives all the surface information and assembles it on the screen.

4. goto 1

The key points to note are that only 2D information is communicated, thus the sever need only store a collection of 2D messages and so will not have a high memory requirement, even though the distributed data set can be huge. In addition, the size of the messages are kept to a minimum, so hopefully a communications bottleneck will not exist. It is worth noting that only server-client communication is required, not client-client. All communication is done using PVM.

1.2. pV3 interface

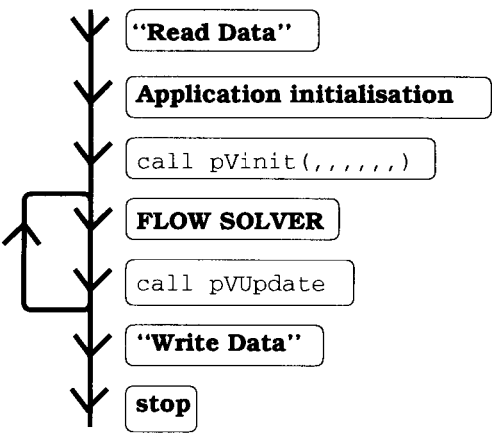


Figure 2. A typical flow diagram for a CFD code

Figure 2 is a schematic of the control flow of a "typical" CFD code, and is given by way of example to show how **pV3** could be used in this case. The box with read data is in quotes since when the code is executing in parallel, this involves receiving the portion of the grid the current processor has been assigned to work on; similarly for the write data box. This will be discussed in more detail in the OPlus section. In this simple example two **pV3** calls are marked, `pV_init(....)` which is called before any iterative flow solver has begun, the arguments of this routine provide the **pV3** library with the following information:

- the cell types used (i.e. tetrahedra, hexahedra, prisms, pyramids or any mixture);

- the type of the grid movement; dynamic (e.g. an adaptive mesh or multigrid application where the connectivity changes), or fixed throughout the calculation;
- the type of data to be viewed, dynamic (changing between each iteration) or fixed;
- the mesh co-ordinates type which also may be dynamic or fixed;
- the number boundary surfaces, e.g. (1) solid wall, (2) free-stream, (3) inlet, and etc.

This information will be known before the calculation proceeds, and in general does not require anything additional to be introduced into the code. From the above list it is also evident that a wide variety of grids (including multiblock, hybrid and adaptive) and applications (including steady and unsteady) can be accommodated.

The second **pV3** call in Figure 2 is to `pV_update`, which has one argument (the simulation time for the current iteration), and is the entry point for the visualisation. Within this library routine, a number of programmer supplied routines may be called, typical **pV3** call-backs are:

`pVGrid` returns the current grid co-ordinates;

`pVCell` returns the current cell connectivity;

`pVSurface` returns boundary surface connectivity;

`pVScal` returns scalar data associated with the grid co-ordinates.

Once the above routines have been written and calls to `pV_init(....)` and `pV_update` have been inserted into the application code, interactive visualisation of distributed data is accomplished. Further work is required by the programmer to enable streamlines to pass between partitions.

2. THE OPlus LIBRARY

All of the parallel solvers described in this paper use the OPlus library [2], (Oxford Parallel Library for Unstructured Solvers). This takes the majority of the parallelising burden away from the application programmer. The key aspects of the library are summarised below:

- The library enables the straightforward parallelisation of a general class of unstructured grid applications, this includes any grid type (tets, hexas, and etc.) and many data structures (cell-based, edge-based, and face-based).
- The library enables an application programmer to develop and maintain parallel code, this is a single-source FORTRAN code, even though the parallel execution uses a master-slave arrangement [4].
- The resulting parallel code has no sends and receives, instead a high level loop syntax is used, where the library performs all the message passing.

- The OPlus philosophy insists on consistent parallel-sequential execution, that is the resulting output data from a parallel or sequential run will be identical to within machine precision. The verification of this correct parallel execution by comparison with sequential execution, is a straightforward and mechanical task.

- One item overlooked by many parallel strategies is input/output, here a consistent parallel-sequential i/o in enforced, i.e., the input and output files from a parallel and sequential execution are identical. Consequently it is trivial to change the number of processors to be used.

- The library automatically partitions, performs local renumbering and schedules communication, without any user interaction.

All the application programmer need do is adopt the OPlus loop syntax and channel the i/o through specific subroutine calls. It is hoped that by using the OPlus system, the effort required to parallelise an application should be similar to that required for vectorisation. A consequence of this is the parallel machine is utilised at an early stage of the code development. That is when the code uses Oplus' loop syntax and input/output structure, it is parallelised. Thus the initial testing of the code on simple problems, which may be very time consuming, can be executed on the parallel machine.

Interfacing **pV3** with OPlus is a straightforward task, and once complete the **pV3** software enables the interruption of any executing code, and the results to be displayed as the computation proceeds. Since **pV3** is interacting with the executing code, not just conserved variables can be displayed, since many other useful quantities are available, such are residuals, limiter values, and etc.

3. EXAMPLES

When this paper was presented at the conference, a short movie was shown, this best exemplified the output and power of **pV3**. In this paper a short summary of the three examples shown in the movie is given.

3.1. Debugging Aide

One example of the debugging uses of **pV3** can be seen during some initial testing of a flow solver on a 3D tetrahedral Ni bump grid, see Fig. 4. The Euler flow solver is an edge based MUSCL scheme with a Roe-type Riemann solver, combined with Runge-Kutta time stepping. The convergence history displayed the residuals decaying by two orders of magnitude then levelling out. When the parallel execution of this was interrupted and interrogated by the **pV3** server, contours of residuals of density revealed disturbances being advected from the leading and trailing edge of the bump, see Fig. 4. It was subsequently easy to identify a limit cycle caused by the limiter used in the CFD code. This is just one example of how parallel machines can be used in a development environment.

3.2. Multigrid

A multigrid algorithm is used to accelerate the convergence of a cell-based Lax-Wendroff Euler flow solver [5], for an aircraft configuration, see Fig. 3. A sequence of non-nested tetrahedral grids is used, the finest of which has 6.7 million tetrahedra, 1 million nodes. All

of the grids need to be partitioned consistently; geometric locality between grids is thought to be an important performance consideration. Consequently the strategy adopted is for the coarse grids to "inherit" the partition of the fine grid. The finest grid is partitioned using inertial geometric bisection. All partitioning was performed automatically by the OPlus library.

This calculation needed 1 Gbyte of storage which in this case was distributed over 8 (IBM) SP1 nodes. The visualisation was performed on an SGI INDIGO (R4400) workstation. This application clearly demonstrates the power and flexibility of **pV3**.

An example of a fine mesh partition around the Nacelle/pylon/wing region, and the subsequent inherited coarse partition can be seen in Fig. 3. It is clear to see how the data is kept local between the grids.

3.3. Unsteady flow

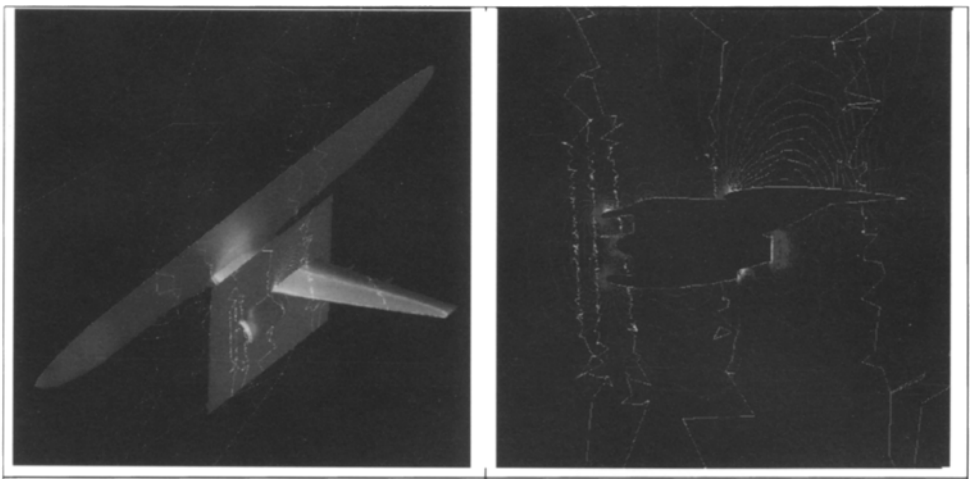
The aforementioned multigrid strategy has been used as in implicit solver for some time accurate calculations [3]. Here the aerodynamic response has been calculated from the periodic pitching of an aircraft. This was modeled using moving meshes, which is an extra degree of complexity which **pV3** can accommodate.

4. CONCLUSIONS

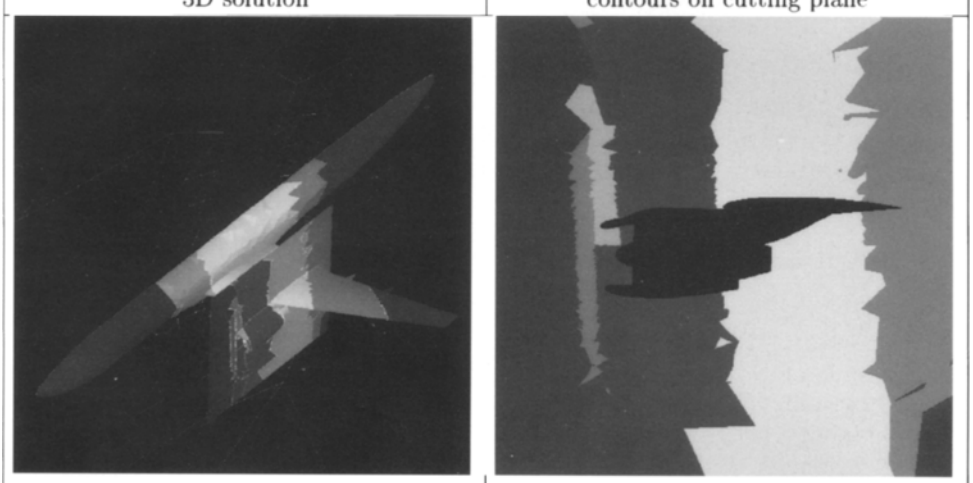
Interactive visualisation of distributed data using **pV3** is a powerful tool for the debugging and development of parallel code, as well as an invaluable aide to the interrogation of large 3D data sets. This can be achieved with minimal programming, using the **pV3** library interface.

REFERENCES

1. D. A. Burgess, P. I. Crumpton, and M. B. Giles. A parallel framework for unstructured mesh solvers. proceedings of ECCOMAS conference on CFD, 1994.
2. D. A. Burgess, P. I. Crumpton, and M. B. Giles. A parallel framework for unstructured mesh solvers. IFIP WG10.3 Working Conference on Programming Environments for Massively Parallel Disributed Systems, 1994.
3. P. I. Crumpton and M. B. Giles. Implicit time accurate solutionson unstructured dynamic grids. *AIAA Paper 95-1671*, 1995.
4. P. I. Crumpton and M. B. Giles. Oplus: A parallel framework for unstructured solvers. *Parallel CFD 95*, This procedings.
5. P.I. Crumpton and M.B. Giles. Aircraft computations using multigrid and an unstructured parallel library. AIAA Paper 95-0210, 1995.
6. R. Haimes. pV3: A distributed system for large scale unsteady CFD visualisation. *AIAA Paper 94-0321*, 1994.
7. R. Haimes. Concurrent distributed visualization and solution steering. *Parallel CFD 95*, This procedings.



3D solution | contours on cutting plane



3D surface partition | partition on cutting plane

Figure 3.

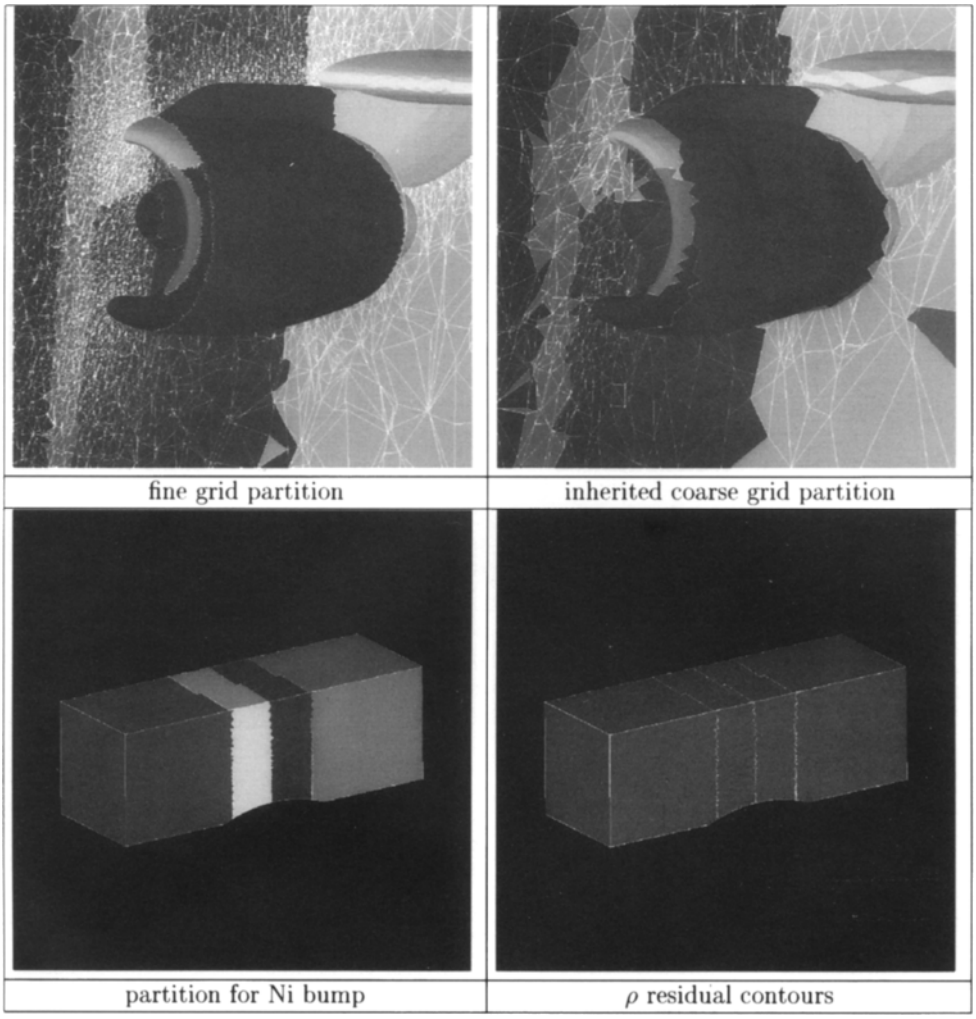

fine grid partition
inherited coarse grid partition
partition for Ni bump
ρ residual contours

Figure 1.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

A Dynamic Load Balancing Technique for Solving Transonic and Supersonic Flows on Networked Workstations

Nobuyuki Satofuka, Masanori Obata and Toshihiro Suzuki

Kyoto Institute of Technology,
Matsugasaki, Sakyo-ku, Kyoto 606, Japan

A Numerical method for solving the Euler equations based on the method of lines has been implemented on networked engineering workstations and multi Transputer system using domain decomposition approach. The message passing is handled by PVM (Parallel Virtual Machine) software. For dynamic load balancing and reduction of computational work a strategy of selective solutions of subdomains based on local residuals has been employed. For steady supersonic and hypersonic flow problems, a reduction of 40~50% computational work and 20~30% of CPU time has been achieved.

1. INTRODUCTION

Today, the use of a network of workstations is considered as an interesting alternative to dedicated parallel computers to enter the world of parallel computing. Since workstations are available with the most modern generation of processor chips, memory is rather cheep and therefore of reasonable size, The cost performance is usually better for workstations than for special parallel computers. For workstations' networks, the speed and the bandwidth of communication are usually significantly inferior to the corresponding value of parallel computers. This implies that much more effort is needed in the construction of algorithms to reduce the amount of communications as far as possible.

During the past years we have been developing a new explicit method for solving the Euler and Navier-Stokes equations on vector computers[1-3]. The approach falls in the category of the method of lines. The governing equations are spatially discretized by appropriate finite-difference approximations. The rational Runge-Kutta scheme[4] is used for the time stepping scheme. The scheme is fully explicit as well as robust even for the stiff problems of high Reynolds number flows. A remarkable improvement of the efficiency is achieved by a combination of the rational Runge-Kutta scheme with several convergence acceleration techniques[3]. The convergence rate to a steady state solution obtained with the scheme may be competitive with

those of implicit approximate factorization schemes for inviscid and viscous flow equations.

In this paper we will present an implementation of our explicit method on networked workstations via domain decomposition. For dynamic load balancing and reduction of computational works, a strategy of selective solution of subdomains based on local residuals has been employed. The performance of the present approach in solving the 2-D Euler equations for supersonic and hypersonic test problems are investigated.

2. EULER EQUATIONS

The two-dimensional Euler equations subject to general coordinate transformation can be written in dimensionless, conservation-law form as

$$\frac{\partial \hat{q}}{\partial t}+\frac{\partial \hat{E}}{\partial \xi}+\frac{\partial \hat{F}}{\partial \eta}=0 \tag{1}$$

where the conservation variables $\hat{q}$ and the flux vectors $\hat{E}$ and $\hat{F}$ are

$$\hat{q}=Jq, \quad \hat{E}=J(\xi_x E+\xi_y F), \quad \hat{F}=J(\eta_x E+\eta_y F) \tag{2}$$

and

$$q=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ e\end{bmatrix}, \quad E=\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ (e+p)u\end{bmatrix}, \quad F=\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ (e+p)v\end{bmatrix} \tag{3}$$

Here ρ is the density, u and v are the Cartesian velocity components, p is the pressure, and e is the total energy per unit volume which is defined for the perfect gas, by

$$e=\frac{p}{\gamma-1}+\frac{1}{2}\rho\left(u^2+v^2\right) \tag{4}$$

The geometrical factors (metrics) ξ_x, ξ_y and so on, are obtained from the derivatives of the Cartesian coordinates of the grid points as

$$\xi_x=J^{-1}y_\eta, \quad \xi_y=-J^{-1}x_\eta, \quad \eta_x=-J^{-1}y_\xi, \quad \eta_y=J^{-1}x_\xi. \tag{5}$$

The transformation Jacobian J is defined by

$$J=x_\xi y_\eta-x_\eta y_\xi \tag{6}$$

Hereafter for simplicity, the symbol ^, which denotes the quantity subject to general coordinate transformation, may be omitted without any confusion, unless the specific note is provided.

3. NUMERICAL PROCEDURE

In our method of lines approach[1-5], the spatial derivatives of the Euler equations (1) are first discretized by the conventional central finite-difference approximation. Then the Navier-Stokes equations become a system of ordinary differential equations:

$$\frac{d}{dt}q_{i,j}=-\frac{E_{i+1,j}-E_{i-1,j}}{2\Delta\xi}-\frac{F_{i,j+1}-F_{i,j-1}}{2\Delta\eta}, \tag{7}$$

where subscript i and j denote the grid indexes such as $q_{i,j} = q(\, i\Delta\xi, j\Delta\eta\,)$. The central finite-difference approximation may produce an oscillatory solution, when the resolution of a computational grid is poor. In such a case, dissipative terms are added in the right hand side of Eq.(7) to eliminate the spurious oscillation. Two types of dissipative models, the scalar dissipation model[5] and the upwind TVD dissipation model[6] are used here.

The dissipative term for ξ- direction can be written as

$$D_\xi = d_{i+1/2,j} - d_{i-1/2,j}.$$

The dissipative flux of the scalar dissipation model is given by the following form,

$$d_{i+1/2,j} = \left[\varepsilon^{(2)}_{i+1/2,j}\Delta_\xi (Jq)_{i,j} - \varepsilon^{(4)}_{i+1/2,j}\Delta^3_\xi (Jq)_{i-1,j}\right] / \left(J\Delta t_\xi\right)_{i+1/2,j}.$$

Here Δ_ξ is the forward difference operator and Δt_ξ is the critical time step which ensures the prescribed local Courant number for ξ-direction. The precise definition of this time step will be mentioned later. The coefficients $\varepsilon^{(2)}_{i+1/2,j}$ and $\varepsilon^{(4)}_{i+1/2,j}$ are defined as

$$\varepsilon^{(2)}_{i+1/2,j} = \omega^{(2)} \max\left(v_{i+1,j}, v_{i,j}\right), \quad \varepsilon^{(4)}_{i+1/2,j} = \max\left(0, \omega^{(4)} - \varepsilon^{(2)}_{i+1/2,j}\right),$$

where $\omega^{(2)}$ and $\omega^{(4)}$ are adaptive coefficients and $v_{i,j}$ is obtained from the second difference of the pressure as:

$$v_{i,j} = \frac{\left|p_{i+1,j} - 2p_{i,j} + p_{i-1,j}\right|}{p_{i+1,j} + 2p_{i,j} + p_{i-1,j}}$$

The discretized Euler equations, Eq. (7), can be rewritten in vector form simply as:

$$\frac{d\vec{q}}{dt} = \vec{Q}(\vec{q}). \tag{8}$$

For the integration of this system of ordinary differential equations, any one of conventional time integration schemes can be used. In this paper, we adopt the 2-stage rational Runge-Kutta (RRK) scheme introduced by Wambecq[4]. When applied to Eq. (8), the scheme may be written in the following form.

$$\vec{g}_1 = \Delta t \vec{Q}\left(\vec{q}^{\,n}\right), \quad \vec{g}_2 = \Delta t \vec{Q}\left(\vec{q}^{\,n} + c_2\vec{g}_1\right), \quad \vec{g}_3 = b_1\vec{g}_1 + b_2\vec{g}_2,$$

$$\vec{q}^{\,n+1} = \vec{q}^{\,n} + \left[2\vec{g}_1(\vec{g}_1, \vec{g}_3) - \vec{g}_3(\vec{g}_1, \vec{g}_1)\right] / (\vec{g}_3, \vec{g}_3),$$

where superscript n denotes the index of time steps and $(\vec{g}_i, \vec{g}_j)$ denotes the scalar product of two vectors $\vec{g}_i$ and $\vec{g}_j$. The coefficients b_1, b_2, and c_2 must satisfy the relations,

$b_1 + b_2 = 1, \quad b_2 c_2 \le -0.5$

An efficient second order scheme is given by the coefficients

$b_1 = 2, \quad b_2 = -1, \quad c_2 = 0.5.$

The time step Δt is determined locally, so that the local Courant number becomes a constant for each grid point. The critical time step for each coordinate direction is obtained in the one dimensional manner as,

$$\left(\Delta t_\xi\right)_{i,j} = \frac{1}{\left(|U| + c\sqrt{\xi_x^2 + \xi_y^2}\right)_{i,j}}.$$

4. PARALLELIZATION

4.1 Parallelization strategy

The numerical scheme described in the previous chapter is implemented and applied for solving transonic and supersonic flows on networked engineering workstations of HP9000 model 720 and multi Transputer system. The message passing is handled by PVM (Parallel Virtual Machine) software. As a parallelization strategy the domain decomposition approach is used in which the whole physical domain is divided into a number of smaller subdomains called blocks. Two lines of overlapping auxiliary grids surrounding a given block is used to make available the information required in the determination of flow variables at the block boundary. Basically each block is treated by a different processor.

4.2 Speedup on two parallel systems

A two-dimensional supersonic duct flow is solved to evaluate the performance of the present approach and to measure the scalability of the parallel systems.

Figure 1 shows the computational domain and the flow conditions. The inlet Mach number is 2.0. The computations are carried out by using two ways of domain decomposition. One uses the computational grid decomposed in ξ-direction, as shown in Fig. 2 (a), and the other uses the computational grid decomposed in η-direction, as shown in Fig.2 (b). The number of grid points is 121x41.

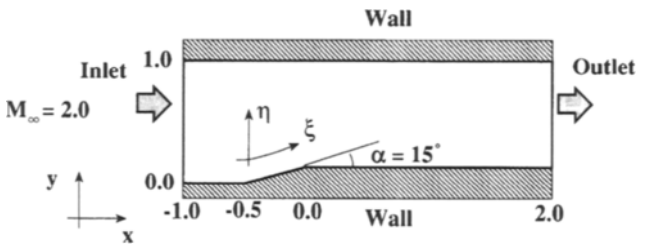
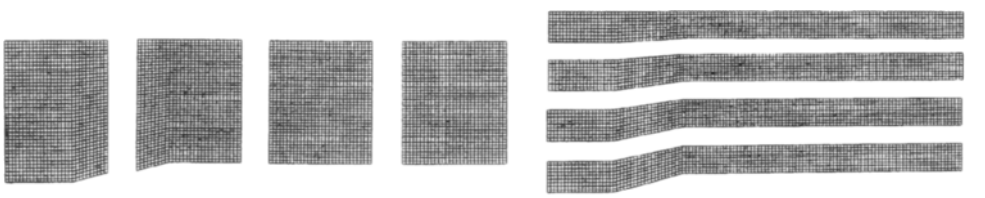


Figure 1 Geometry of two-dimensional duct flow

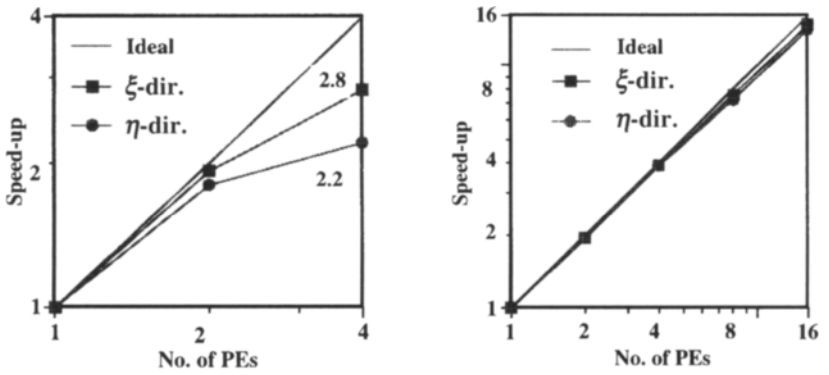


(a) Decomposition in ξ-direction (b) Decomposition in η-direction

Figure 2 Computational grid for two-dimensional duct flow

Figure 3 (a) and (b) show the measured speedup on the network of four workstations and the multi Transputer system with 16 PEs.

Because of high latency and slow communication speed in the networked workstations, the speedup with 4 PEs obtained by using decomposition in ξ-direction is 2.8, as shown in Fig. 3 (a). Since the amount of data to be transferred in η-direction is lager than that for ξ-direction, the speedup for decomposition in η-direction is 2.2. The speedup on the multi Transputer system is almost linear owing to their fast independent serial links.



(a) Networked workstations (b) Multi Transputer system

Figure 3 Measured speedup for two-dimensional duct flow

5. REDUCTION OF COMPUTATIONAL WORK BASED ON LOCAL RESIDUAL

5.1 Basic concept

In the present paper, we propose a strategy to reduce the computational work by selective solutions of subdomains based on local residuals. During the numerical integration of the Euler equations the residuals of each subdomain are periodically checked. When residuals of a certain subdomain drop below a prescribed limit, computation of the subdomain is stopped. When a large number of subdomains become inactive, the loads are redistributed among the available processors.

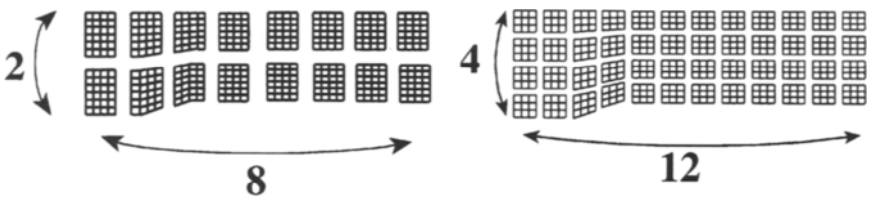
5.2 Application to two-dimensional supersonic duct flow

As a first test case for evaluating the present approach, computation is carried out for two-dimensional supersonic duct flow mentioned in the previous section. The computational grid is divided into 1x1, 8x2 and 12x4 subdomains for ξ- and η-direction, shown in Fig. 4 (a) and (b). As a measure to check the efficacy of the present approach for the reduction of computational work, the work unit is defined as,

$$[\text{Work unit}] = \sum \frac{[\text{No. of Grid Points Computed}]}{[\text{Toal No. of Grid Points}]}.$$

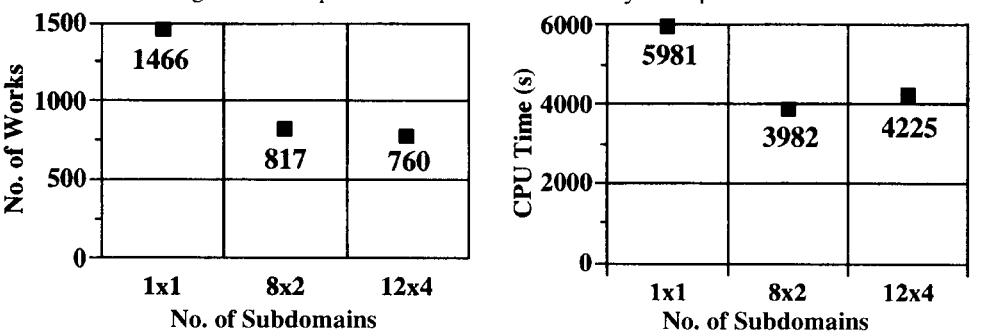
Figure 5 (a) and (b) shows the measured number of work units and CPU time to convergence,

respectively. It is observed that the larger number of subdomains, the less the total amount of work units to convergence. The CPU time for 12x4 subdomains is increased, in comparison with that of 8x2 subdomains, due to the computational overhead of the present approach.



(a) 8x2 subdomains (b) 12x4 subdomains

Figure 4. Computational domain divided in ξ- and η-direction



(a) Work units (a) CPU time (s)

Figure 5. Reduction of work units and CPU time by present method

5.3 Application to Hypersonic flow past double ellipse

As the next example, we apply the present method to a hypersonic flow past a double ellipse at the Mach number M=8.15 and the angle of attack α=30°. Figure 6 shows the computational grid divided into 8x3 subdomains with 257x97 grid points. Table 1 shows the comparison of the work units and CPU time between 1x1 and 8x3 subdomains. In the present approach, it is observed that the total work units for 8x3 subdomains is reduced to 54% of that of 1x1 and the CPU time is reduced to 81%.

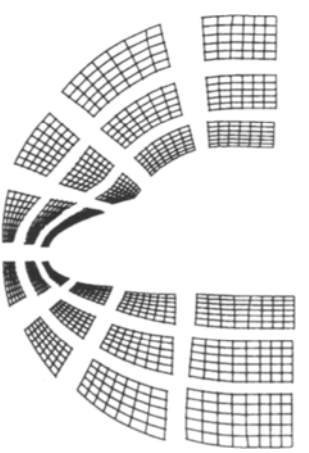


Figure 6 Computational Grid divided into 8x3 subdomains

Table 1. Comparison of work units and CPU time

Domain	Work	CPU Time (s)
1x1	4018	16915
8x3	2156	13705

6. DYNAMIC LOAD BALANCING

Finally, we will apply the same technique mentioned in the previous chapter to dynamic load balancing for networked workstations. For simplicity, we will explain the technique for re-partitioning of the subdomains based on local residuals in the case of two processing elements. Figure 7 shows the procedure. Firstly, the entire computational domain is divided into a larger number of subdomains than the number of PEs. The subdomains are distributed equally to PEs, and each PE begins the flow computation for a steady flow problem. When the temporary convergence criterion is satisfied on a certain number of subdomains, the remaining subdomains are re-partitioned equally to PEs. After the temporary convergence criterion is satisfied on every subdomain, the next temporary convergence criterion is set, and this is repeated until steady state.

Table 2 shows the comparison of CPU time and speedup between the present dynamic load balancing technique and the static load balancing technique in which equal number of grid points is partitioned to each processor at all time. The CPU time of the present approach is less than that of static load balancing, but speedup is 2.54 due to the overhead in the present dynamic load balancing technique.

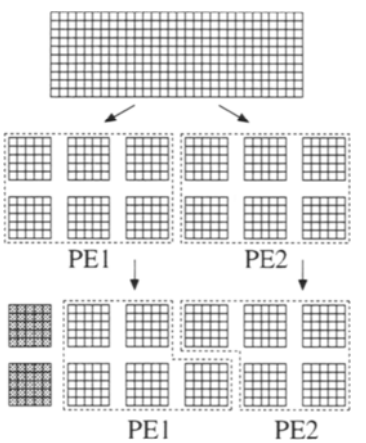


Figure 7. Re-distributed subdomains

Table 2. Comparison of CPU time and speedup

No. of PEs	Dynamic Load Balancing		Static Load Balancing	
	CPU Time(s)	Speedup	CPU Time(s)	Speedup
1	4225	1.00	5043	1.00
2	2412	1.75	2725	1.85
4	1663	2.54	1681	3.00

7. CONCLUSIONS

A Numerical method based on the method of lines has been implemented on workstation network and multi Transputer system using domain decomposition approach. For dynamic local balancing and reduction of computational work a strategy of selective solutions of subdomains based on local residuals has been employed. For steady supersonic and hypersonic flow problems, a reduction of 40%~50% computational work and 20%~30% CPU time has been achieved. For transonic case, virtually no reduction is observed by the present approach.

REFERENCES

1. N. Satofuka, K. Morinishi and Y. Nishida, "Numerical Solution of Two-Dimensional Compressible Navier-Stokes Equations Using Rational Runge-Kutta Method.", Numerical Fluid Mechanics 18, pp. 200-218 (1987)

2. N. Satofuka and K. Morinishi, "Solution of Compressible Euler Flows Using Rational Runge-Kutta Time Stepping Scheme", Numerical Fluid Mechanics 26, pp. 309-330 (1989)

3. K. Morinishi and N. Satofuka, "Convergence Acceleration of the Rational Runge-Kutta Scheme for the Euler and Navier-Stokes Equations", Computers and Fluids Vol.19 No.3/4, pp.305-313 (1991), Technical Papers ISCFD Nagoya, pp. 131-136 (1989)

4. A. Wambecq, "Rational Runge-Kutta Methods for Solving System of Ordinary Differential Equations", Computing 20, pp. 333-342 (1978)

5. A. Jameson and T.J. Baker, "Solutions of the Euler Equations for Complex Configurations", AIAA Paper 83-1929 (1983)

6. H.C. Yee, "Upwind and Symmetric Shock-Capturing Schemes", NASA TM89464 (1987)

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Scalable Parallel Generation of Partitioned, Unstructured Meshes

D.C. Hodgson, P.K. Jimack, P. Selwood and M. Berzins

School of Computer Studies, University of Leeds, Leeds LS2 9JT, UK

In this paper we are concerned with the parallel generation of unstructured meshes for use in the finite element solution of computational fluid dynamics problems on parallel distributed memory computers. The use of unstructured meshes allows the straightforward representation of geometrically complicated domains and is ideally suited for adaptive solution techniques provided the meshes are sensibly distributed across the processors. We describe an algorithm which generates well-partitioned unstructured grids in parallel and then discuss the quality of this mesh and its partition, and how this quality can be maintained as the mesh is modified adaptively.

1. INTRODUCTION

The usual approach to solving finite element (or finite volume) problems in parallel on a distributed memory machine is to decompose the mesh into a number of subdomains and to allocate each of these subdomains to a processor. This decomposition of the elements of the mesh should have two main features. Each subdomain should contain approximately the same number of node points (or elements for cell-centered finite volumes), so as to achieve "load-balancing". Also, the number of node points (or edges for cell centered finite volumes) which lie on the boundary between different subdomains (we will refer to such points as "interpartition boundary vertices/edges") should be kept to a minimum since the amount of interprocessor communication will depend upon this number.

There has been a considerable amount of research into the problem of partitioning an existing mesh across distributed memory in a manner compatible with the above (see for example [4,9] and references therein). However these methods assume that the mesh is not held in a distributed manner across a multi-processor machine but is stored in one place. This is clearly prohibitive since the size of the mesh is constrained by the memory available on the single processor on which it is stored. In addition, if we wish to solve very large problems in parallel we do not want the mesh generation and partitioning to be a serial bottleneck. For these reasons the main contribution of this paper is to illustrate a technique for generating an automatically *partitioned* mesh in *parallel*. This technique is outlined in the next section and its performance is analyzed and discussed in section 3.

In section 4, a number of extensions of the work are considered, including its application to time-dependent problems using adaptivity through local h-refinement and derefinement. Here being able to generate a well-partitioned initial mesh in parallel is not sufficient since the refinement process will destroy the load-balance as time progresses. The issue of dynamically modifying the existing partition is therefore addressed.

2. THE PARALLEL MESH GENERATOR

In order to create a mesh in parallel, the domain must be split up into subregions which can then be meshed simultaneously and independently. Our method does this by producing in serial an initial coarse, or background, triangulation (tetrahedralization in 3-d) of the domain, using a Delaunay algorithm as described in [14], and then distributing this background grid across the processors in an intelligent manner before meshing begins.

This distribution of the background grid is performed so as to ensure that each processor will generate a mesh of about the same size *and* the number of interpartition boundary vertices will be very low. In order to achieve this, a weighted dual graph of the background grid is first produced, as shown in figure 1. Here, the weight of each graph vertex is equal to the number of nodes that it is estimated will be generated within the background element to which that vertex corresponds ($W_{v(\ell)}$ say), and the weight of each graph edge is equal to the number of nodes that it is estimated will be generated along each background edge (face in 3-d) to which that graph edge corresponds ($W_{e(n,m)}$ say). This weighted dual graph is then partitioned using a recursive spectral bisection algorithm, such as described in ([3,9]). Spectral algorithms seek to partition the graph so that the number of boundary vertices is minimized subject to the constraint of maintaining load-balance. This means that, provided our estimates, $W_{v(\ell)}$ and $W_{e(n,m)}$, are reliable, the partition of the background grid should be very well-suited for the parallel generation of our unstructured mesh.

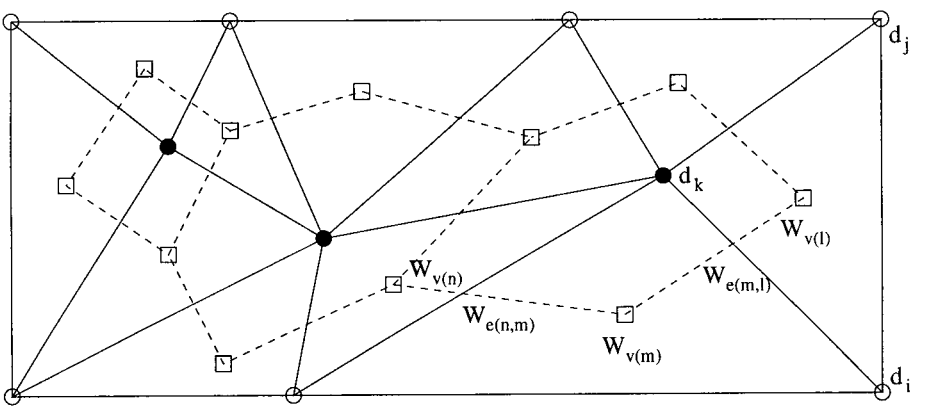


Figure 1. An example of a coarse background grid in 2-d along with its weighted dual graph (with vertex weights $W_{v(\cdot)}$ and edge weights $W_{e(\cdot,\cdot)}$). Also shown are some typical point distribution values (d_i, d_j and d_k).

The mesh generation itself also uses the same Delaunay algorithm [14], this time in parallel on each processor. Two options are available: either each processor can Delaunay mesh the union of those background elements which it has been allocated, or it can mesh

each of its background elements separately. In either case, the density of the generated mesh is governed locally through the use of point distribution values at each coarse grid vertex. These allow different target mesh spacings to be specified throughout the domain, as described in [14]. An additional, and equally important, use for these point distribution values is to allow the estimates $W_{v(\ell)}$ and $W_{e(n,m)}$ to be formed very cheaply using straightforward geometric formulae. For example, in two dimensions, the simple estimate

$$W_{v(\ell)} = \frac{\text{Area of Element } \ell}{(\frac{d_i+d_j+d_k}{3})^2}$$

proves to be surprisingly reliable, where the point distribution values, d_i, d_j and d_k, are shown in figure 1 (see [5] for further details).

This mesh generation procedure appears to be both well load-balanced and highly scalable. The load balance comes from the fact that each processor is generating a mesh contribution of about the same size, even on a highly irregular mesh, whilst the scalability comes from the fact that the only sequential steps are applied to the coarse background grid rather than the final mesh itself. Moreover, once the background grid has been partitioned there is no need for any inter-processor communication to take place (providing consistent algorithms are used to mesh subdomain boundaries so as to ensure that they match-up on neighbouring processors). In contrast with this, the methods described in [1] and [6] both farm out subregions to processors for meshing, without distributing them in a considered manner. This means that the generated mesh may not be very well balanced across the processors and that there is no guarantee that subregions sharing a processor will be connected. It is therefore always necessary to repartition these meshes once they have been generated in parallel. On the other hand, the parallel generator suggested in [7] *is* designed to produced meshes that are already well partitioned, using a "wavefront" approach to split up a background grid. However, no attention is explicitly paid to keeping the number of interpartition boundary vertices low, so the quality of the partitions produced is likely to be affected by this.

3. PERFORMANCE

The algorithm described in the previous section has been implemented using MPI ([8]) so as to ensure portability. It has been run successfully on a variety of platforms, including a distributed memory computer (a 64 node Intel hypercube), a shared memory computer (an 8 processor SGI Power Challenge) and also on a cluster of 16 SGI Indy workstations.

The outcomes of some typical computations are shown in Figure 2 and Table 1. Here, an unstructured mesh has been generated around a NACA0012 aerofoil using a point distribution function which is suitable for a supersonic flow (free stream Mach number = 2.0) with a moderate Reynolds number ($Re = 500$). The coarse background grid contains 1239 elements and the mesh that is generated contains almost half a million elements. The generation times for this mesh were 62.2 seconds on the Intel i860 (using 8 processors) and 40.2 seconds on a cluster of 8 Indy workstations. As can be seen, the mesh density varies enormously throughout the domain, yet each partition is of a very similar size. Demonstrating that the number of interpartition boundary vertices is also low is quite hard to do quantitatively for an example of this size, however one can see from the shaded coarse mesh in Figure 2 that the subdomains are all connected and have compact shapes.

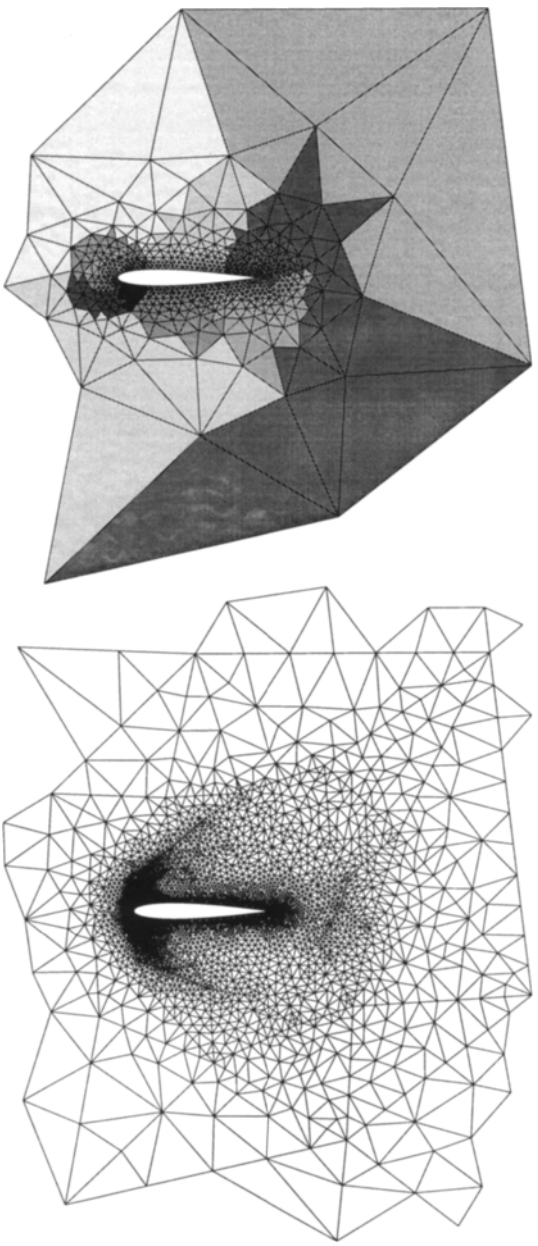


Figure 2. The coarse background mesh (with shading to illustrate how it has been parti tioned into 8 subregions) and the final mesh generated around a NACA0012 aerofoil.

Table 1
Details of the parallel generation of a 447 403 element unstructured mesh around a NACA0012 aerofoil.

Subdomain	Vts.	%age diff.	i860	Indys
1	28639	+0.6	62.2	40.2
2	28061	-1.4	51.1	31.3
3	28928	+1.6	62.1	35.2
4	27925	-1.9	51.7	29.1
5	28552	+0.3	47.2	21.4
6	28215	-0.9	52.8	27.7
7	28763	+1.1	50.6	25.3
8	28613	+0.5	50.2	22.1

Experiments with larger meshes have shown that as the number of processors and the mesh size increase the generator can be shown to scale reasonably well. For example, using 16 nodes on the Intel i860 to generate a mesh in excess of a million elements takes little over 100 seconds. Also, the maximum difference in the size of each subregion always appears to be between -5% and $+5\%$ of the average size.

The generation of separate Delaunay meshes within each element of the background grid means that the final mesh is only locally Delaunay. This does not appear to affect the quality of this mesh adversely however since the number of coarse elements is always far smaller than the number of elements actually generated.

It is important to stress that, even though the number of vertices generated by each processor is about the same, the time taken by each processor varies more greatly. This is entirely due to the fact that the mesh being generated is of such a variable density, which causes some processors to be allocated many fewer background elements than others. It is these processors which are the last to finish since the sequential generation algorithm used, [14], is slower at generating a few large meshes than a large number of moderate meshes. Ideally therefore, the density of the coarse mesh should be made to reflect the required final mesh density everywhere. When this is done, the variation in meshing times between processors falls of drastically (thus leading to greater efficiency).

Another improvement that can be made to the algorithm as it is described above, is to attempt to reduce further the small (5%) variations in the mesh sizes on each processor. Such variations can still lead to a noticeable drop in the efficiency of a parallel solver, and so a small amount of post-processing of the partition may be worthwhile. One solution is to move a background element (and its sub-mesh) from one processor to another which contains fewer nodes, while keeping the total number of interpartion boundary vertices as low as possible.

4. EXTENSION TO ADAPTIVE SOLUTION METHODS

Generation of the initial fine mesh is only one part of the solution process. The use of adaptive methods means that the initial mesh will have to be both refined and coarsened

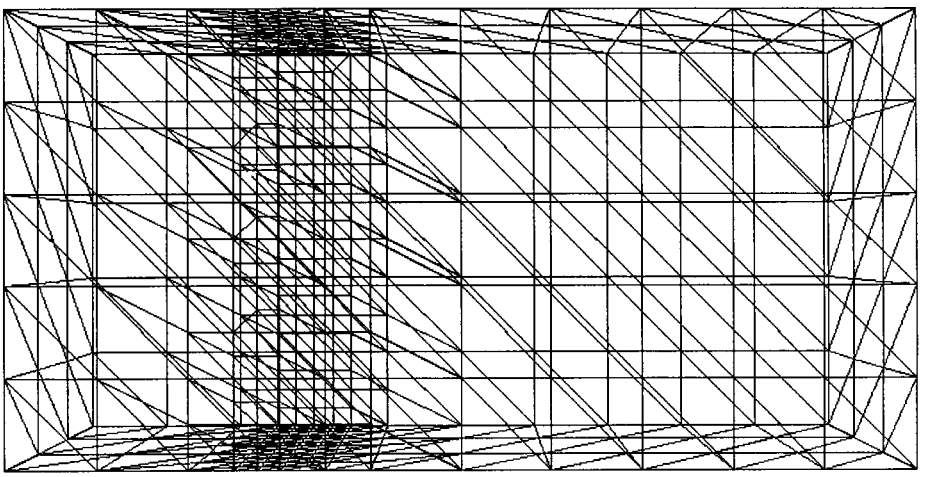


Figure 3. An unstructured 3-d mesh which accurately represents a steep front which is about to be advected from left to right.

in different areas on the basis of computed error estimates. This problem of dynamic load balancing during the adaptive solution of time-dependent problems has been considered by a number of authors, such as [2,11,12]. This problem is essentially the same as that already described above: where an existing partition has a low number of interpartition boundary vertices but a less than perfect load balance.

The problem is illustrated by Figure 3 which shows a three dimensional tetrahedral mesh, generated using the code of [10], that is used to represent a simple function with a shock. This is used as initial data for the linear advection equation, with the shock being advected to the right. In [10] an unstructured mesh adaption algorithm is described which is suitable for just this class of problem. As the front advects, the mesh is refined immediately ahead of it and coarsened immediately behind it. A key issue for any parallel solver is therefore to ensure that as the mesh changes, each processor maintains a roughly equal share of the elements and unknowns. One way to achieve this would be to generate an entirely new mesh every few time-steps, using the technique of Section 2. Alternatively, one could use conventional hierarchical refinement and derefinement but form a new partition of the mesh whenever adaptivity has occurred. Both of these approaches seem inefficient however, as they fail to make the best use of the existing partition.

The approach here is to make use of hierarchical refinement and derefinement of a coarse background grid, and to only consider altering the partition of this background grid after each adaptive step has occurred. As with the original mesh generation, this has the advantage that the partitioning problems considered are always much smaller than if one were to attempt to directly partition the mesh at its finest level. Moreover, it is possible to use repartioning techniques such as those provided by the software package JOSTLE ([13]), to ensure that the partition before refinement is used as the basis for the

Total number of		% Imbalance	
Adaptivity cycles	Timesteps	Before Repartitioning	After Repartitioning
0	0	-	2.22
5	15	25.67	2.9
10	30	10.97	0.84
15	45	19.56	2.69
20	60	15.98	1.46

Table 2
Partition imbalance for a mesh adapting to follow a shock

partition after refinement whenever possible. This has the further advantage that most of the background elements (and therefore the data for the sub-meshes within them) will remain on the same processor as they were before the adaptive step.

As an example of this, the mesh shown in Figure 3, which contains two levels of refinement beneath the background grid, is partitioned equally across 4 processors. As the solution evolves, the load balance of the initial partition is lost, as some of the coarse elements (to the right) refine further, whilst others derefine. Table 2 shows how the imbalance in the partitioning occurs as the solution evolves. Note that by using JOSTLE to repartition the weighted dual graph of the background grid, we can regain the balance, and hence the efficiency, of the partition. Moreover, the vast majority of the coarse elements (and their sub-mesh data) remain in the same memory locations as before the repartitioning.

This simple three-dimensional example again demonstrates that the approach of working mainly with the weighted dual graph of a background grid appears to have significant potential. There are still a number of issues associated with this which should be addressed, but the underlying approach seems to be both efficient and effective.

There are two main difficulties which are currently being investigated further. Firstly, after a very large number of adaptive steps it may be necessary to discard the present partition altogether and repartition the problem from scratch. This is most likely to occur if the background grid is excessively coarse or if extremely high levels of refinement are being used in small, localized regions. The other issue is that of whether the local repartitioning of the coarse background mesh can itself be efficiently implemented in parallel which is desirable from a scalability point of view.

ACKNOWLEDGEMENTS

We would like to thank Philip Capon, Peter Dew, Bill Speares, Nasir Touheed and Chris Walshaw whose contributions and suggestions have all been most valuable. We are also grateful to Shell Research for funding and supporting the development of the 3-d adaptive code. The work of the first and third authors is funded by the EPSRC (under a postgraduate studentship and grant GR/J84919 respectively), and those calculations performed on the Intel i860 made use of the machine and support provided by Daresbury

Laboratory, UK (under EPSRC grant GR/J27066).

REFERENCES

1. T. Arthur, M.J. Bockelie. *A Comparison of Using APPL and PVM for a Parallel Implementation of an Unstructured Grid Generation Program.* Tech. Report 191425, NASA Computer Sciences Corporation, Hampton, Virginia, 1993.
2. P. Diniz, S. Plimpton, B. Hendrickson, R. Leland *Parallel Algorithms for Dynamically Partitioning Unstructured Grids.* In Proc. of the Seventh SIAM Conference on Parallel Processing for Scientific Computing, ed. D.H. Bailey *et al* (SIAM), pp. 615 - 620, 1995.
3. B. Hendrickson, R. Leland. *An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations.* SIAM Jour. on Sci. Comp., Vol. 16, No. 2, pp 452-469, 1993
4. D.C. Hodgson, P.K. Jimack. *Efficient Mesh Partitioning for Parallel P.D.E. Solvers on Distributed Memory Machines.* In Proc. of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, ed. R.F. Sincovec *et al* (SIAM), pp. 962 - 970, 1993.
5. D.C. Hodgson, P.K. Jimack. *Parallel Generation of Partitioned, Unstructured Meshes.* In Advances in Parallel and Vector Processing for Structural Mechanics, ed. B.H.V. Topping and M. Papadrakakis (Civil-Comp Press), pp. 147–157, 1994.
6. A.I. Khan, B.H.V. Topping. *Parallel Adaptive Mesh Generation.* Computing Systems in Engineering, Vol. 2, No. 1, pp. 75-101, 1991.
7. R. Löhner, J. Camberos, M. Merriam. *Parallel Unstructured Grid Generation.* Computer Methods in Apl. Mech. Eng., 95, pp. 343-357, 1992.
8. Message Passing Interface Forum. *MPI: A Message Passing Interface standard.* Int. J. of Supercomputer Applications, 8, 1994.
9. H.D. Simon. *Partitioning of Unstructured Problems for Parallel Processing.* Computing Systems in Engineering, Vol. 2, No. 2/3, pp. 135-148, 1991.
10. W. Speares, M. Berzins. *A Fast 3-D Unstructured Mesh Adaption Algorithm with Time-Dependent Upwind Euler Shock Diffraction Calculations.* In Proceedings of the 6th International Symposium on Computational Fluid Dynamics, pp 1191-1188, 1995.
11. A. Vidwans, Y. Kallinderis, V. Venkatakrishnan. *Parallel Dynamic Load-Balancing Algorithm for Three-Dimensional Adaptive Unstructured Grids.* AIAA Journal, Vol. 32, No. 3, pp. 497-505, 1994.
12. C.H. Walshaw, M. Berzins. *Dynamic Load-Balancing for PDE Solvers on Adaptive Unstructured Meshes.* Concurrency: Practice & Experience
13. C.H. Walshaw, M. Cross, S. Johnson, M.G. Everett. *JOSTLE: Partitioning of Unstructured Meshes for Massively Parallel Machines.* To appear in Proceedings of Parallel CFD '94, Kyoto.
14. N.P. Weatherill, O. Hassan. *Compressible Flowfield Solutions with Unstructured Grids Generated by Delauney Triangulation.* AIAA Journal, Vol. 33 No. 7, pp 1196-1204, 1995.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Evaluation of the JOSTLE mesh partitioning code for practical multiphysics applications

K. McManus*, C. Walshaw*, M. Cross, P. Leggett, and S. Johnson

Parallel Processing Group, Centre for Numerical Modelling & Process Analysis,
University of Greenwich, London, SE18 6PF.
email: [k.mcmanus, ..]@gre.ac.uk ; URL: http://www.gre.ac.uk/~[k.mcmanus, ..]

The use of unstructured mesh codes on parallel machines is one of the most effective ways to solve large computational mechanics problems. Completely general geometries and complex behaviour can be modelled and, in principle, the inherent sparsity of many such problems can be exploited to obtain excellent parallel efficiences. However, unlike their structured counterparts, the problem of distributing the mesh across the memory of the machine, whilst minimising the amount of interprocessor communication, must be carefully addressed. This process is an overhead that is not incurred by a serial code, but is shown to be rapidly computable at run time and tailored for the machine being used.

1. INTRODUCTION

Multiphysics simulations integrate the solution of interacting physical processes to solve complex inhomogeneous models, such as, metals casting and aeroelasticity. The University of Greenwich is developing a three dimensional unstructured mesh code, PHYSICA [3], which brings together into one toolkit the modelling of many processes such as:

turbulent multiphase fluid flow
phase changes (i.e. melting/solidification)
free surface flows
fluid-structure interaction
magnetohydrodynamics
elasto-viscoplasticity
structural dynamics
contact analysis

This code is being parallelised for Distributed Memory Multi Instruction Multi Data (DM MIMD) architectures using explicit message passing in Fortran77 [5].

Partitioning of an unstructured mesh into P partitions that are mapped to P processors is well known to be NP complete. Many methods have been developed that partition a graph corresponding to the communication requirements of the mesh. A new method for solving this graph-partitioning problem has been devised at the University of Greenwich and encapsulated in a software tool, JOSTLE [7]. It employs a combination of techniques

*Sponsored by the Engineering and Physical Science Research Council.

to give a rapid initial partition together with a clustering technique to further speed up the process. The resulting partitioning method is designed to work efficiently in parallel as well as sequentially and can be applied to both static and dynamically refined meshes. In this paper we present results obtained by the JOSTLE procedure for parallel multiphysics applications on unstructured meshes.

2. THE JOSTLE MESH-PARTITIONING CODE

The underlying strategy of the JOSTLE code is based on the continuing trends of research issues and computing resources. As mesh and machine sizes grow, the need for parallel load-balancing becomes increasingly acute. For small meshes (N nodes) and small machines (P processors), an order N overhead for the mesh partitioning may be considered reasonable. However, for large N and P, this order of overhead will rapidly become unacceptable if the solver is running at $O(N/P)$.

In addition, it is often the case that the mesh is already distributed across the memory of the parallel machine. For example, parallel mesh generation codes or solvers which use parallel adaptive refinement give rise to such distributed meshes, and in these cases it is extremely expensive to transfer the whole mesh back to a single processor for sequential load-balancing, if indeed the memory of that processor allows it.

To tackle these issues efficiently, the strategy developed here is to derive a partition as quickly and cheaply as possible, distribute the mesh and then *optimise* the partition *in parallel*. If the mesh is already distributed then the existing partition is used and optimisation can commence immediately. Experiments, on graphs with up to a million nodes, indicate that the JOSTLE procedure is up to an order of magnitude faster than existing state-of-the-art techniques such as Multilevel Recursive Spectral Bisection [1].

2.1. Topology mapping

A pertinent but often ignored factor in parallel processing is the underlying topology of the machine's interconnection network. Even on machines with small numbers of processors, it is possible to detect variations between the latencies of processors which are closely linked and those which are 'far apart'. Although most machines now have facilities for passing messages between two non-adjacent processors without interrupting intermediate processors, high contention of the interprocessor links can result if adjacent partitions are mapped to, say, opposite corners of a processor array. As the trend towards massively parallel machines continues, these effects are likely to be exacerbated and machine topologies will have an increasingly important effect on the parallel overhead arising from any given partition. Most of the current generation of mesh partitioning algorithms, however, take no account of the machine topology. The mapping to the machine is either treated as a post-processing step, applied after the data has been partitioned, or even ignored. For machines with small numbers of processors this may be a legitimate simplification, but as machine sizes increase it is likely that a poor mapping will cause significant performance degradation.

We use an undirected graph $G(N, E)$, of N nodes & E edges, to represent the data dependencies arising from the unstructured mesh. Any partition of G produces a graph S describing sub-domain connectivity and loosely the mapping problem can be thought of as the placing of this S onto the processor topology such that the communication overhead

is minimised. Figure 1 shows three possible partitions of a mesh along with the resulting sub-domain graphs S. We concentrate here on mapping onto a grid topology where we assume that the processors are connected as a 1D, 2D or 3D array. This is a realistic restriction as grids can be found in some of the current range of parallel machines such as the Intel Paragon (2D) or Cray T3D (3D).

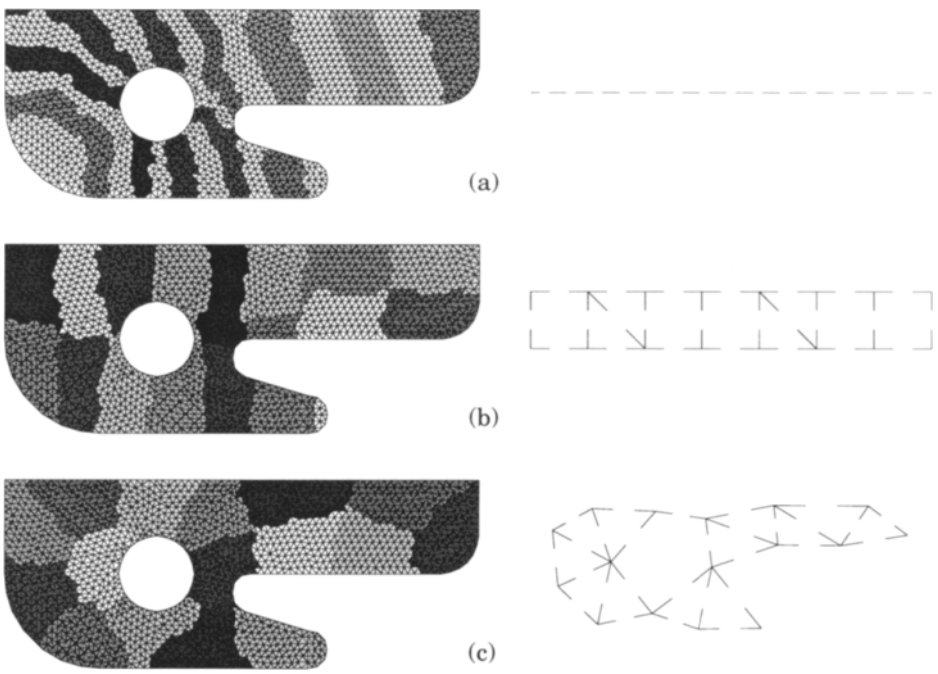


Figure 1. Partitions of a 2D mesh into (a) 1D, (b) 2D and (c) uniform topologies with the corresponding sub-domain connectivity graphs.

2.2. The initial partition

The aim of the initial partitioning is to divide up the graph as rapidly as possible prior to optimisation, where most of the work takes place. We use two different initial partitioning algorithms; the Greedy Algorithm ignores the processor topology completely, whilst the other, Geometric sorting, does a very crude mapping onto a processor grid.

The Greedy algorithm used here is a simple variant of that originally proposed by Farhat and fully described in [4]. This is clearly seen to be the fastest *graph-based* method as it only visits each graph edge once. However, it takes no account of the processor topology. The variant employed here differs from that proposed by Farhat in that it works solely with a graph rather than the nodes and elements of a finite element mesh.

Geometric sorting is a simple and intuitive algorithm which partitions solely on the

geometric coordinates of the nodes. Thus, to map a graph onto an $p \times q$ processor grid (where $p \geq q$) the nodes are first sorted by x-coordinate, say, and split into p sets each of weight N/p. The nodes of each of these sets are then sorted by y-coordinate and split into sets of N/pq. Of course, neglecting connectivity information may result in a very poor quality partition and/or mapping. However if nodes which are adjacent in the graph are also adjacent geometrically, as is frequently the case in graphs arising from finite element/finite volume discretisations, it can be very successful.

2.3. Optimisation methods

The two optimisation methods outlined here have different aims; *uniform optimisation* treats the processor topology as uniform and tries to minimise the number of interprocessor cut-edges. *Grid optimisation*, on the other hand, treats the processor topology as a grid and attempts to optimise the mapping by eliminating non-local communications.

The uniform optimisation algorithm is fully described in [6] where it is seen that a key part of the technique is the way in which each sub-domain tries to minimise its own surface energy. In the physical 2D or 3D world the object with the smallest surface to volume ratio is the circle or sphere. Thus the idea behind the sub-domain heuristic is to determine the centre of each sub-domain (in some graph sense) and to then measure the radial distance from the centre to the edges and attempt to minimise this by migrating nodes which are furthest from the centre. The code finally decides which nodes to migrate based on a combination of radial distance, load-imbalance and the change in cut-edges.

The grid optimisation algorithm is based very much on the uniform optimisation algorithm with some minor changes and a more appropriate method for minimising the surface energy. After some experimentation it was found that using the radial distance as a basis for migrating nodes which are far from the sub-domain centre was simply not appropriate for achieving a grid mapping, as nodes which are relatively far away from the centre of the sub-domain may be well placed for the topology mapping. To see this, consider a partition for a 1D processor array as in Figure 1(a) where the partition preserving the topology is just a series of strips. Migrating nodes which are far away from the centre of the sub-domain (i.e. at the extremes of each strip) does not preserve the partition as a 1D array. If however, we attempt to minimise the width of each strip, rather than the radial distance, we do find that the partition can preserve the machine topology. Thus, instead of measuring the radial distance of the sub-domain, we measure (in a graph sense) the distance between the borders with processor on the left and the processor on the right. This technique can be extended to higher dimensional arrays by each processor classifying the other processors as lying, in the 2D case, to either the north, south, east or west, with processors lying on a diagonal falling into two sets [6].

2.4. Mapping strategies

Table 1 describes the four mapping strategies tested. The *unmapped* partition completely ignores the processor topology to give a near optimal partition for a uniform topology as in Figure 1(c). The *postmapped* partition is the unmapped partition remapped to the processors with a processor allocation algorithm applied post-partitioning. This algorithm continually swaps sub-domains between processors until no further improvement in the map cost is possible. The *premapped* partitioning method works the other way round; the graph is initially mapped, albeit crudely, onto the processor grid and then

Table 1
Mapping strategies

Strategy	Initial partition	Optimisation	Processor allocation
Unmapped	Greedy	Uniform	No
Postmapped	Greedy	Uniform	Yes
Premapped	Geometric sort	Uniform	No
Mapped	Geometric sort	Grid	No

optimised to minimise the number of interprocessor cut-edges. Because the final partition does not deviate far from the initial partition the resulting sub-domain graph still 'fits' reasonably well onto the processor grid. Indeed, although processor allocation was not used for these results, in tests it was very rare that it could find better allocations. Finally the *partition mapping* strategy acknowledges the processor topology throughout as in Figure 1(b).

3. PARALLEL PHYSICA

Research into multiphysics modelling by the Greenwich group has led to the specification and development of PHYSICA, a modelling software framework for multiphysics phenomena. The core component of PHYSICA is a code structure which provides a three dimensional unstructured mesh framework for the solution of any set of coupled partial differential equations up to second order.

For Finite Volume (FV) procedures the evaluation of fluxes across cell/element faces, volume sources, and coefficients of the linear solvers in the iterative procedures is generic, being essentially based upon mesh geometry and material properties within a cell. As such, the code can be structured so that nodes, FV cell faces and cell volumes can all be calculated automatically and considered as software objects. Since, nodes, cell faces and volumes are all considered as objects the mesh can be conceived of as simply the tool for providing information on the connectivity of nodes, cell faces and volumes; its description may be structured as such in a memory management system which has been designed so that it makes no presuppositions on the geometric structure of the cells. Given that the representation of the mesh connectivity is described by a memory management system, it is straightforward to extend it to include a database system for the storage of all the run time information as well as for model results of any given run. All equation solvers are generic and constructed so that they may be called interchangeably by the user with consistent data structures.

PHYSICA provides a SIMPLE based solution procedure for the fully compressible Navier-Stokes equations with all variables co-located at cell centres using a modified Rhie-Chow method to estimate velocities at cell faces, plus a number of differencing methods to specify the convection terms. A range of turbulence models including k-ϵ, RNG k-ϵ and other length scale techniques together with enthalpy based solidification/melting procedures are coupled with the fluid flow solver. Cell centred elastoviscoplastic solid mechanics with contact analysis are coupled within the false time stepping.

The JOSTLE code is integrated into a PHYSICA prototype to provide at run time a

partition of the mesh elements. Face-based and a node-based partitions are derived from the element partition to fully decompose the mesh into sub-domains. Each sub-domain is extended with a surface of elements, faces and nodes overlapping the neighbouring sub-domains. These overlaps carry variables required for the solution of the variables within the sub-domain. Variables in the overlaps are updated from the the processors on which the variable is calculated [5]. A consequence of this sub-domain extension is to increase the sub-domain connectivity. Sub-domains that were not connected in the original partition may become connected through the overlaps. Consider the sub-domain graph in Figure 1(b), here the maximum node degree is four. After applying overlaps to the sub domains the maximum node degree increases to five.

4. RESULTS

The test case used is a solidification problem solving flow, heat and stress over a 60,000 element mesh. This problem was run on the University's Transtech Paramid machine. This machine has 28 i860XP based processing nodes, each of which is equipped with 32 or 16Mbyte of fast DRAM and a T800 communication co-processor. The processor nodes are hard connected in pairs with Inmos C004 multi-stage crossbar switches providing interconnection between the node pairs. This configuration allows great versatility in node interconnection topology. An obvious and simple arrangement for the Paramid topology is a $p \times 2$ grid which is the arrangement used for these results. A virtual channel router resident on each node allows message passing between all of the nodes on the machine as though the machine were a fully connected network.

In the following two graphs the solid lines refer to partitions reflecting a $p \times 2$ processor topology and the dashed line indicates a partition reflecting a 1D pipeline or $p \times 1$ topology.

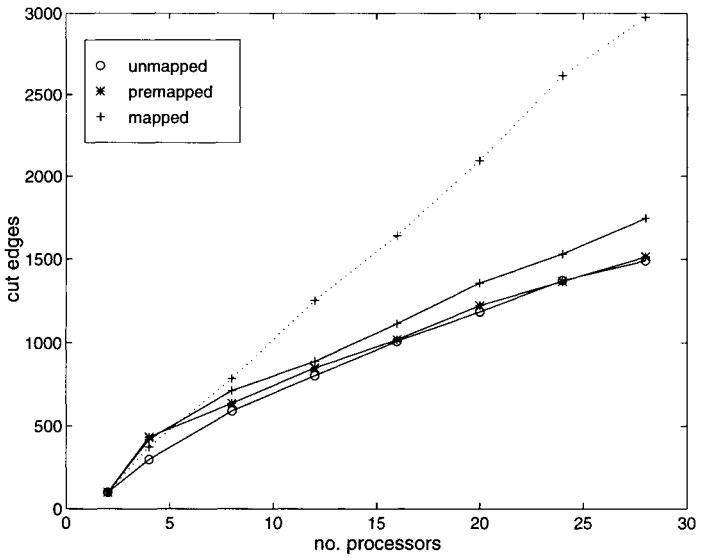


Figure 2. Number of cut edges for a range of partitions.

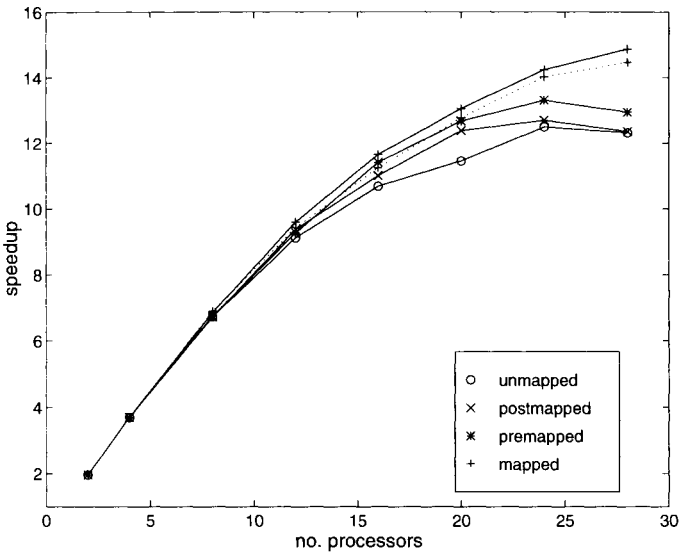


Figure 3. Speedup for a range of partitions on an i860 based Transtech Paramid.

The lowest number of cut edges is given by the unmapped (postmapped) partition but this does not give the best speedup performance. The unmapped and postmapped partitions are actually the same; however the postmapped has in addition an optimised mapping of partitions to processors applied to it. Where the two partitions give a similar speedup this reflects an unintentionally fortuitous mapping of the unmapped partition. It is possible that the unmapped and postmapped partitions may by chance be identical, it is however highly unlikely that the unmapped partition would ever give a better speedup than the postmapped partition. The best overall speedup performance is given by the mapped partitions, despite the cut edge count being higher than the other partitions. This confirms our proposition that partitioning in accordance with the machine topology will result in improved performance. Using a 1D partition leads to a significantly higher number of cut edges and consequently the message length is far greater, however fewer messages are required. In this case only two latencies are required for each overlap update which explains the unexpectedly good speedup results for the pipeline partition. Given that the imbalance of elements between the the sub-domains is less than 0.25% it is apparent from this result that the machine performance with this code is latency bound.

Start-up latency on the Paramid has been measured as $33\mu s$ with a peak bandwidth of 1.7Mbytes per second. This bandwidth is not sustained with virtual channel routing and degrades to around 1.3 for near neighbour communication and can get as low as 0.9 for non local messages. This can deteriorate further to around 0.3Mbytes per second if the communication channels are saturated. While this bandwidth is low in comparison with other parallel machines [2] the latency is reasonably good. Similar performance may therefore be expected from other platforms.

Partitioning onto a $p \times q$ processor array where $q > 2$ has yet to be tested, but is not expected to improve performance on the Paramid because of the latency bound. Whilst

a $q = 2$ mapped partition is likely to incur five latencies, a $q > 2$ mapped partition can incur eight latencies, but will not significantly reduce the number of cut edges until P increases considerably.

5. MACHINE TOPOLOGY PROFILE

In spite of what parallel machine manufacturers may claim there will always be a distance related communication cost. This cost becomes more significant as the number of processors increases. To quantify the variations in latency and bandwidth we have developed a code which measures the communication performance of a parallel machine. Latency is measured by the simple method of sending a short message between each processor on the parallel machine. Similarly bandwidth is measured by sending a large message between each processor. These measurements are initially carried out with only one message being passed at any one time, and then with every node communicating simultaneously. This provides a peak and a saturated performance measure that may be expressed as a weighted graph (matrix) that describes the communication performance between each pair of processors. What is immediately apparent is the non-uniform performance described by the graph. Such a weighted graph can be obtained quickly, at run time, and then used by the partitioning code to ensure that the mesh partition produced is appropriate for the measured machine communication profile as opposed to a notional topology that may not be reflected in actual communication performance. It is anticipated that this scheme will provide improved performance across a range of parallel machines without the need to understand or specify the architecture of the machine.

REFERENCES

1. S. T. Barnard and H. D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice & Experience*, 6(2):101–117, 1994.
2. J. Dongarra and T. Dunigan. Message passing performance of various computers. Tr, Oak Ridge National Laboratory, University of Tenessee and Oak Ridge National Laboratory, 1995.
3. M. Cross et al. Towards an integrated control volume unstructured mesh code for the simulation of all of the macroscopic processes involved in shape casting. *Num. Meth. Industrial Forming Processes (NUMIFORM 92)*, pages 787–792, 1992.
4. C. Farhat. A Simple and Efficient Automatic FEM Domain Decomposer. *Comp. & Struct.*, 28:579–602, 1988.
5. K. McManus, M. Cross, and S. Johnson. Integrated Flow and Stress using an Unstructured Mesh on Distributed Memory Parallel Systems. In *Parallel CFD'94*. Elsevier, 1995. (in press).
6. C. Walshaw, M. Cross, and M. Everett. A Localised Algorithm for Optimising Unstructured Mesh Partitions. *Int. J. Supercomputer Applications*, 1995. (in press).
7. C. Walshaw, M. Cross, S. Johnson, and M. Everett. JOSTLE: Partitioning of Unstructured Meshes for Massively Parallel Machines. In *Parallel CFD'94*. Elsevier, 1995. (in press).

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)
1995 Elsevier Science B.V.

PARALLEL COMPUTATION OF UNSTEADY SUPERSONIC-INLET FLOWS

Suresh C. Khandelwal[a], Gary Cole[b], and Joongkee Chung[c]

[a]NYMA Inc., NASA Lewis Research Center group, Brook Park, OH 44142

[b]NASA Lewis Research Center, Cleveland, OH 44135

[c]ICOMP, NASA Lewis Research Center, Cleveland, OH 44135

1. ABSTRACT

One of the goals of controls/computational fluid dynamics (CFD) interdisciplinary research at NASA Lewis is to develop time-accurate CFD codes which can provide faster turnaround times within an integrated analysis/design system. Since parallel execution can provide faster turnaround times, a two-fold objective was pursued: (1) the conversion of an existing serial code, NPARC-3D (version 2.0), into a coarse grain parallel code based on block functionality, and (2) the use of this parallel NPARC code to investigate the unsteady inviscid flowfield characteristics of a Mach 2.5, variable diameter centerbody (VDC) inlet. The parallel code was tested on both a network of workstations and on the Cray T3D. The speedup relative to a single processor was 14 to 14.5 on 16 networked RS6000-590 workstations and 14.5 to 15 on the Cray T3D. The turnaround time for the complete solution of an unsteady flow problem using the parallel code was six times faster when executing in parallel on the 16 RS6000-590 workstations as compared with the execution serially on the Cray Y-MP8. Full details of the code's performance and a discussion of the unsteady flowfield results are presented in this paper.

2. INTRODUCTION

As part of the High Performance Computing and Communication Program (HPCCP), NASA Lewis is conducting a controls/CFD interdisciplinary research program [1]. The objective of this program is to enable controls engineers to use CFD simulations as numerical test beds for obtaining high-fidelity system models for control system design, testing, and validation. Because of their long execution times, CFD simulations have been impractical for application to control problems. Therefore, a primary goal of this program is to develop faster, optimized time-accurate CFD codes. The research is initially being used to investigate control problems for inlets tytpical of the NASA High Speed Research (HSR) program.

A Euler/Navier-Stokes CFD code, NPARC [2,3], was used to perform a simulation involving an unsteady inviscid inlet flow. The specific simulation, a VDC inlet, has a rather complex geometry which NPARC can handle by defining the overall geometry as a series of blocks. Parallel or distributed computing in a networked environment allows the blocks to be processed independently in parallel. If the code is efficiently parallelized, such a technique can significantly reduce the turnaround time of an application and allow for the computation of complex applications in a reasonable amount of time. The parallel code's performance and the results for the unsteady flow calculations are discussed in this paper.

3. HARDWARE

The code was run on three different platforms—a Cray YMP 8E/8128, a Cray T3D, and the Lewis Advanced Cluster Environment (LACE). The Cray YMP 8E/8128 has 8 processors with 128 Mwords of shared memory running under UNICOS 8.0.2.3. Execution time on single Cray YMP processor was used as a benchmark for comparison. The Cray T3D has 64 processors running under UNICOS-MAX 1.0.2. Each processing element has a local memory of 8 Mwords. All I/O is performed through the attached Cray Y-MP8. The LACE consists of 32 networked IBM RS6000 workstations—16 Model 590 and 16 Model 560. The network has two modes of communication: Ethernet and an IBM ALLNODE switch which runs under two modes—Internet Protocol (IP) and Low Latency Peripheral Interface (LLPI). The Ethernet mode of communication is basically a single, serial connection which must be shared by all workstations. The ALLNODE switch allows multiple, simultaneous connections to exist between workstations. All 590 models are connected via one ALLNODE switch, and all 560 models via another ALLNODE switch. All RS6000 units run under AIX 3.2.5 and have at minimum 128 Mbytes of memory. In addition to having a local file system on each processor, an NFS file server available on LACE can be accessed by all 32 processors.

4. CFD CODE

The NPARC serial code allows for the simulation of steady-state and transient flows, using either viscous or inviscid flowfield calculations. For this study, a number of modifications were made to NPARC (version 2.0) to enhance and facilitate time-accurate calculations including: (a) specification of uniform time step size over all blocks [4], (b) specification of upstream (free-stream) and downstream (exit) boundary conditions as functions of time, and (c) the ability to output variables in "real" (as opposed to normalized) time. In addition, a new exit boundary condition [5] was added to accurately represent the interface to an engine (compressor-face) corrected air-flow demand.

Execution of the code can be broadly divided into three steps: initialization, solution, and output. During initialization, all the gradients, conservation variables (density, momentum, and energy), and block-interface data are calculated and stored. The solution

step is an iterative process performed using two loops. The outer loop counts the number of iterations until convergence conditions are met. The inner loop counts the number of blocks as defined in the input data, reads block data for each iteration, and performs calculations before writing the block data back to the same file. All block-interface data are also written to a file. After the iterative process is complete, flowfield output is also written to a file. Except for common interfaces, the blocks are independent of each other, and this independence was exploited to parallelize the code. As initialization and output are performed only once, the solution step is the main time-consuming element of this algorithm because it executes all the blocks sequentially during each iteration. For a typical CFD problem, thousands of iterations are carried out to arrive at the final solution.

The parallel algorithm implemented is based on the single program multiple data (SPMD) paradigm where one block of data is executed by one process created under the Unix environment with all processes synchronized at the end of each iteration. The block-interface data are handled either via message-passing on an available network or reading/writing to a common file on an NFS file system (file I/O). To get reproducible data, it was necessary to open a file for each process at the start of an iteration and then close it at the end. The sequential I/O on an NFS file system was obtained by allowing only one process to read/write at a time.

Under the multitasking Unix environment on LACE, the parallel code can be executed using any number of processors ranging from a single processor to one for every block defined in the problem. When computing problems with a number of different block sizes, load balance on the system is achieved by manually assigning more than one process to a processor. However, in computing a problem outside a multitasking environment, as with the Cray T3D, the number of processors must equal the number of blocks.

5. MESSAGE-PASSING LIBRARIES

5.1 Application Parallel Portable Library

The Application Parallel Portable Library (APPL) [6] was designed to use a portable message-passing paradigm which allows an application to be developed once and then ported to a variety of parallel platforms. The parallel NPARC code was run under the APPL software environment on the LACE cluster.

5.2 Parallel Virtual Machine (PVM)

PVM [7] was used to run the code on the Cray T3D. The PVM routines support the communication calls. Interface software developed at NASA Lewis was used to provide an APPL-compatible interface to the Cray PVM calls.

6. ENGINE INLET APPLICATION

The geometry chosen for this study is based on a bicone mixed-compression supersonic inlet designated as the VDC [8]. Two three-dimensional grids were generated, one a "fine" grid of 50,950 points and the other a "coarse" grid of 25,104 points. Both grids are divided into 16 blocks, each having nearly the same number of points. The average number of grid points per block is 3,184 for the fine grid and 1,569 for the coarse grid. Both grids have 18 block interfaces. (See Figure 1.) The inlet has three supporting struts (120 degrees apart) blocking the passage in the diffuser. Since calculations were made for a zero-degree angle of attack, only a 60-degree section was needed for the computation. Only inviscid flowfield calculations were made.

7. RESULTS

7.1 Benchmark Runs

The serial code was run on a variety of hardware to obtain baseline performance and to determine relative speeds. It can be executed by keeping all block data either in the memory or on a file. The elapsed time was obtained when all block data were memory resident. When simulating the fine grid geometry, the serial code on the Cray Y-MP8 ran two-and-a-half times faster than the Model 590, five times faster than the Model 560, and about nine times faster than a single processor of the Cray T3D. When simulating the coarse grid geometry, the serial code ran about twice as fast on the Cray Y-MP8 than the Model 590, four times faster than the Model 560, and about eight times faster than a single processor of the Cray T3D (Figure 2).

Timing data were collected over 1,000 iterations to obtain the parallel code speedup. A speedup is defined as the ratio of the elapsed time in serial execution to the elapsed time in parallel execution. For speedup calculations, only the time elapsed during the execution of the solution algorithm is used.

On 16 RS6000 processors with ALLNODE in IP mode, the elapsed time (in sec.) with the fine grid data was 115.7 for the Model 590 and 376.2 for the Model 560, compared to 1,673 for the Model 590 and 3,078 for the Model 560 with the serial code. This corresponds to a speedup of 14.5 on the Model 590 and 8.2 on the Model 560 with ALLNODE. The elapsed time (in sec.) on 8 RS6000 processors was 205.7 with a speedup of 8.1 on the Model 590 and 610.0 with a speedup of 5.1 on the Model 560 (Figure 2). The greater than expected speedup of 8.1 could possibly be attributed to memory caching. With the coarse grid, on 16 RS6000 processors, elapsed time (in sec.) was 51.7 with a speedup of 14.1 on the Model 590 and 149.3 with a speedup of 8.8 on the Model 560. Elapsed time with the serial code was 726.9 and 1,312.6 for both the Models 590 and 560, respectively. The elapsed time (in sec.) on 8 RS6000 processors was 94.6 with a speedup of 7.7 on the Model 590 and 197.5 with a speedup of 6.7 on the Model 560 (Figure 2). Use of 8 processors provided better efficiency for both the fine and coarse grids, most probably due to less communication overhead.

On the Cray T3D, elapsed time with the parallel code in sec. was 382.0 for fine grid and 182.4 for coarse grid data; for the serial code it was 5,703.0 and 2,741.7, respectively. This corresponds to a speedup of 14.9 for the fine grid and 15.0 for the coarse grid.

I/O on the Cray T3D is not efficient because it is handled by the system-load dependent Cray Y-MP8. The effect of I/O is more visible when the serial code is executed on one processor, keeping the block data on a file where it is read and later written to the same file for each iteration, than when all block data are stored in memory. For example, for the coarse grid, the serial code required 5,703.0 sec. to complete 100 iterations using file I/O compared to 277 sec. for the memory-resident case.

When ALLNODE and Ethernet network performance was compared on LACE, ALLNODE provided the better performance, about four times as fast as Ethernet when simulating the coarse grid and about twice as fast when simulating the fine grid. Performance was slightly better in IP than LLPI mode (Figure 3).

The availability of a fast network on a workstation cluster helps to obtain satisfactory speedups only for a well-balanced problem where all the blocks are the same size. When all the blocks are not the same size, parallel execution via file I/O (where all inter-block data were handled via an NFS file system) compares favorably with the message-passing option because, in this case, execution is controlled by the process with the biggest block. Slower/faster network communication is masked under waiting time by the processes as they are synchronized at the end of each iteration. For a well-balanced problem, the file I/O option is much slower than the message-passing option.

7.2 Time-accurate Flow Perturbation Studies

The parallel code performance was also investigated by running practical cases that are typical of those of interest to controls analysts. In order to design a control system, the engineer needs to know the time response between various input/output variables. For this study static-pressure responses in the subsonic portion of the VDC inlet duct were obtained for a step change in either free-stream temperature or exit (compressor-face) Mach number. The time-accurate calculations were performed in three steps: (1) the steady state solution was obtained by using time steps of 2.0E-06 sec. for the fine grid and 4.0E-06 sec. for the coarse grid with Mach numbers of 2.5 for the free stream and 0.33 for the compressor-face; (2) after achieving steady state, the code was executed for 0.01 sec. without changes in input data; and (3) the code was executed for another 0.06 sec. after a perturbation was made—either a 3% reduction in compressor-face Mach number or a 4% increase in the free stream temperature. In both cases the effect on the static pressure due to the perturbation from the steady state condition was calculated. The Runga-Kutta 3-stage solver was used for these calculations.

7.3 Convergence and Perturbation Studies

The residual values are obtained to check the output compatibility of the parallel code relative to the serial code and agree very well for both the coarse and fine grids (Figures 4 and 5). To reach steady state, 20,000 and 35,000 iterations were needed for the coarse and fine grids, respectively. Convergence of residual values took the longest time in blocks 10 to 13 (see Figures 4 and 5 for coarse and fine grid convergence patterns). At

steady state, the axial location of the terminal shock was at 3.387 (ratio of distance from centerbody tip to cowl-lip radius) for the coarse grid and at 3.440 for the fine grid densities. A difference in shock location is expected due to the difference in grid densities.

The transient behavior of the static pressure due to perturbation in the compressor-face Mach number or free stream temperature was evaluated for both grids. Comparisons of results obtained in block 10 at axial locations 3.945 for the coarse grid and at 3.947 for the fine grid are shown in Figure 6. The transient pressure results obtained with the parallel code agreed with the serial code results from the Cray Y-MP8. Results obtained with the fine grid are considered to be more accurate and agreed much better with the results from the 2D code [5] than did the coarse-grid results. This is unfortunate since the fine grid calculations take approximately four times longer than the coarse grid, and emphasizes the need for further increase in speedup. Some possibilities for increasing speedup include: increasing the number of blocks and processors (although the communication overhead may increase); changes to the CFD (NPARC) solver algorithm [5]; and expected future increases in processor speed.

7.4 Comparison of CPU and Turnaround Times

For a total of 70,000 iterations on the dedicated LACE cluster, the CPU time and elapsed time for the fine grid geometry were 1:55:54 (HR:MIN:SEC) and 2:06:28, respectively, using 16 Model 590 processors with ALLNODE in IP mode. Running the serial code on the Cray Y-MP8 (not dedicated), CPU and elapsed times were 13:04:58 and 49:44:25, respectively. Since elapsed time on the Cray Y-MP8 is a function of the system load, CPU time can be used as the best estimate of elapsed time as on a dedicated system. Results can be obtained six times faster with parallel code on LACE than with serial code on Cray.

Similar results were obtained with the coarse grid for a total of 37,500 iterations on the dedicated LACE cluster. The CPU and elapsed times using the parallel code were 0:27:47 and 0:32:39, respectively. On the Cray Y-MP8 with the serial code these times were 3:39:34 and 5:43:42. A comparison of elapsed time on LACE and CPU time on the Cray Y-MP8 shows that the turnaround time on LACE is at least six times faster.

8. CONCLUDING REMARKS

The serial NPARC code was successfully parallelized to run in SPMD mode using the APPL library. This parallel NPARC code performed well in a network environment, is compatible with the serial code, and can provide faster turnaround time under a distributed computing environment. Speedups achieved with the parallel code are encouraging and routine use of CFD simulations for controls/CFD interdisciplinary studies is coming closer to reality. Achieving even faster turnaround times is required. This can possibly be achieved by a combination of more blocks and processors, a faster time-accurate solver in NPARC, and faster processors.

REFERENCES

1. Cole, G., et. al., "Computational Methods for HSCT-Inlet Controls/CFD Interdisciplinary Research," AIAA 94-3209, June 1994.

2. Sirbaugh, J., et al., *A USER'S GUIDE TO NPARC VERSION 2.0*, November 1, 1994.

3. Cooper, G.K., and J.R. Sirbaugh, "PARC Code: Theory and Usage," AEDC-TR-89-15, 1989.

4. Chung, J., "Numerical Simulation of a Mixed-Compression Supersonic Inlet Flow," AIAA 94-0583, Jan. 1994.

5. Chung, J., and G. Cole, "Comparison of Compressor Face Boundary Conditions for Unsteady CFD Simulations of Supersonic Inlets," AIAA 95-2627, July 1995.

6. Quealy, A., G. Cole, and R. Blech, "Portable Programming on Parallel/Networked Computers Using the Application Portable Parallel Library (APPL)," NASA Technical Memorandum 106238, 1993.

7. Sunderam, V., "PVM: A Framework for Parallel Distributed Computing," *Concurrency: Practice & Experience*, Vol. 2, No. 4, December 1990.

8. Saunders, J.D., et. al., "VDC Inlet Experimental Results, First NASA/Industry High Speed Research Propulsion/Airframe Integration Workshop," October 1993.

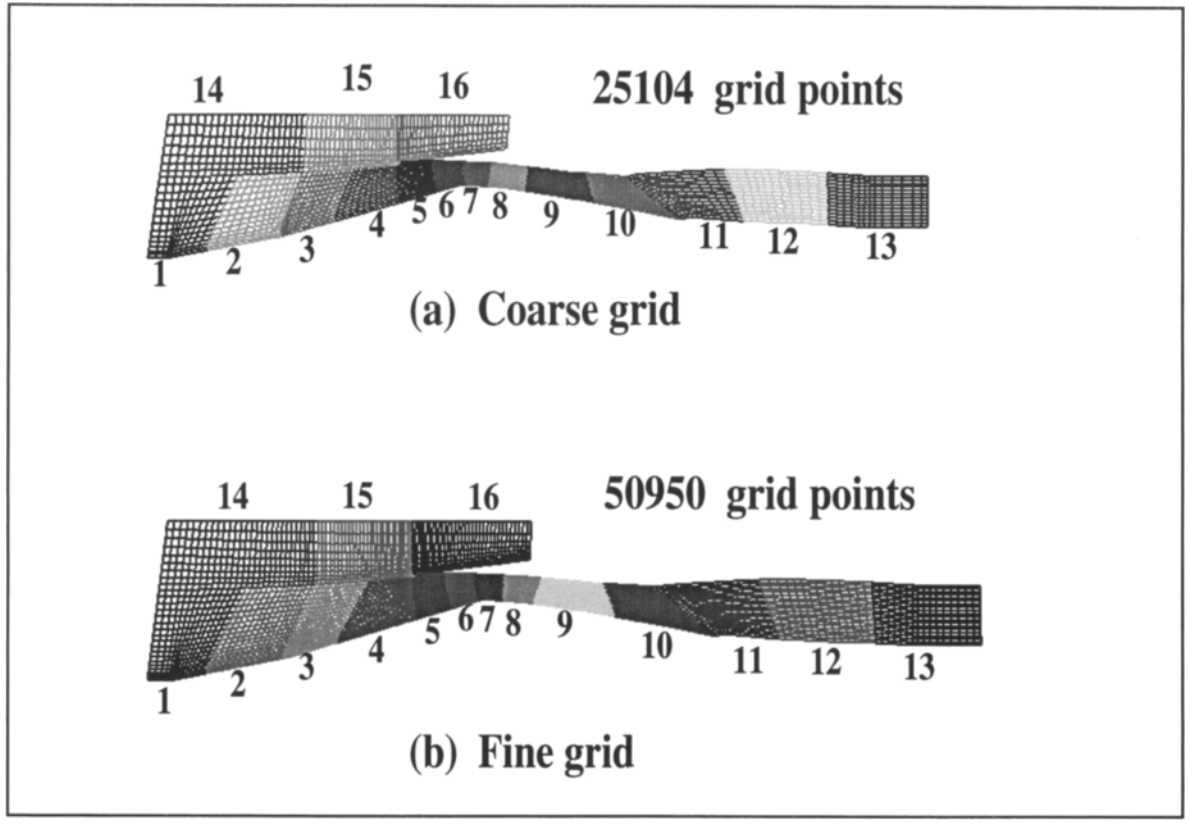


Figure 1. Grids used for a VDC engine inlet study

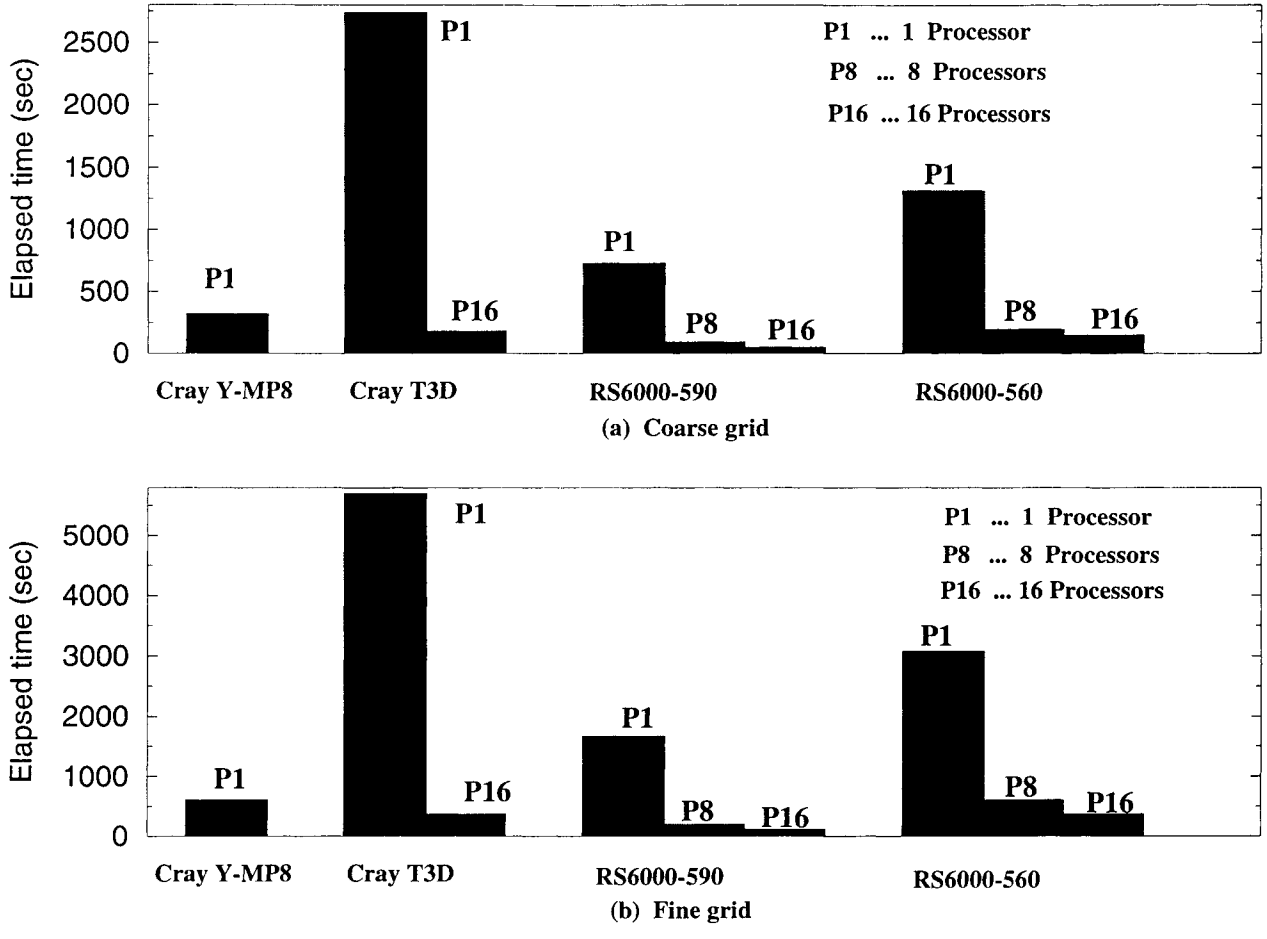
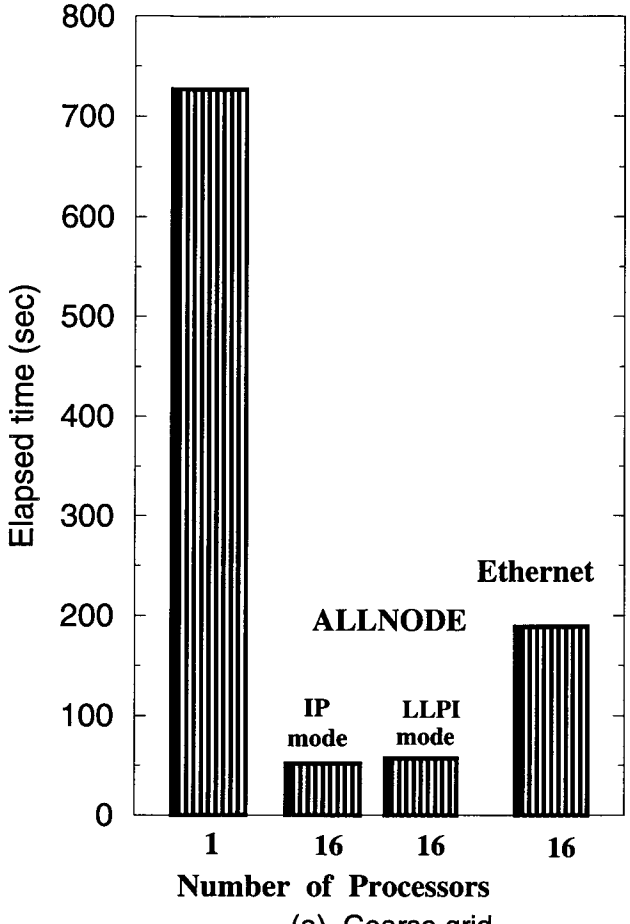
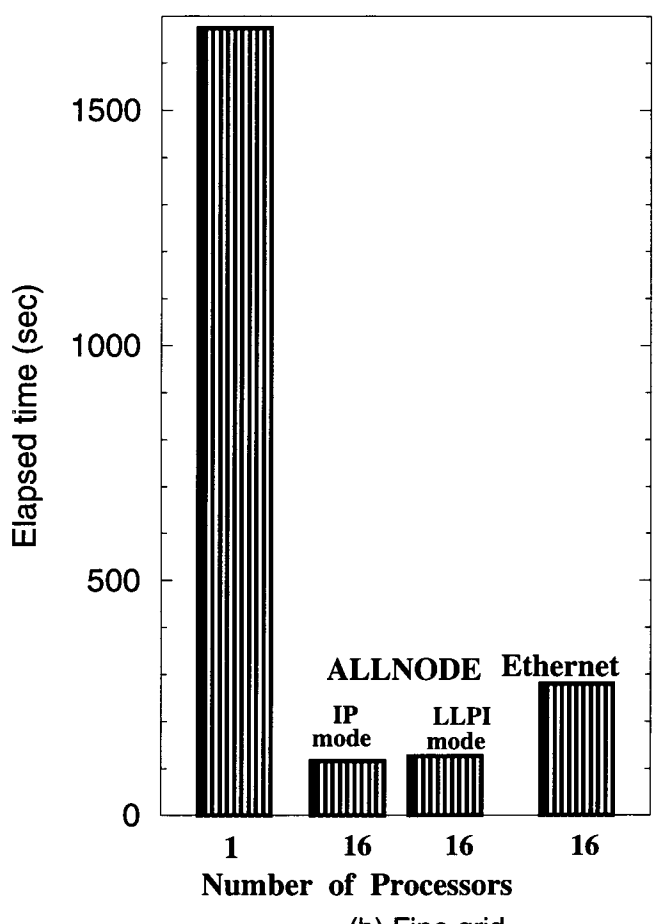


Figure 2. Relative performance of serial and parallel codes



(a) Coarse grid



(b) Fine grid

Figure 3. Network performance on LACE (RS6000-590)

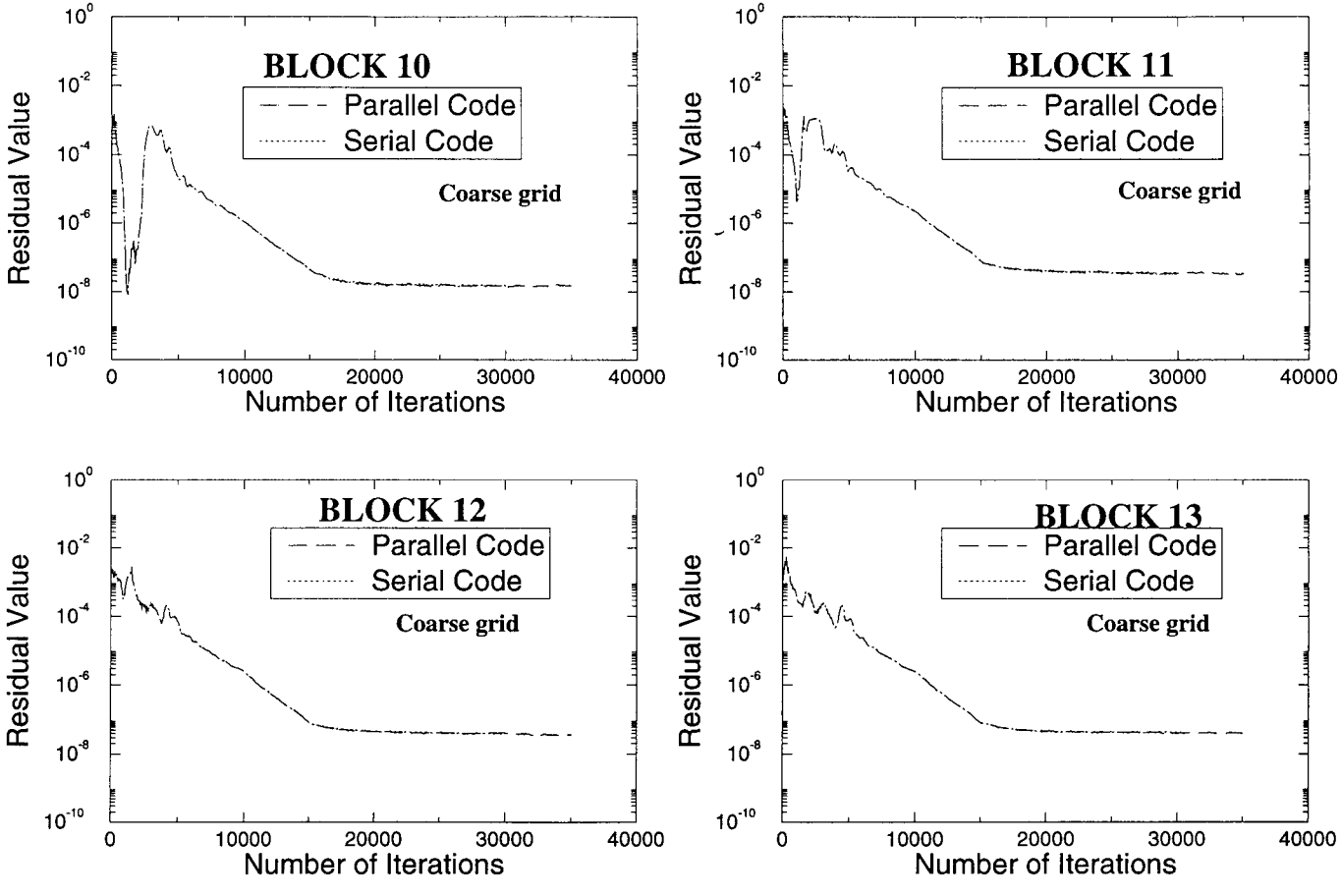


Figure 4. A plot of residual value versus number of iterations using parallel and serial codes

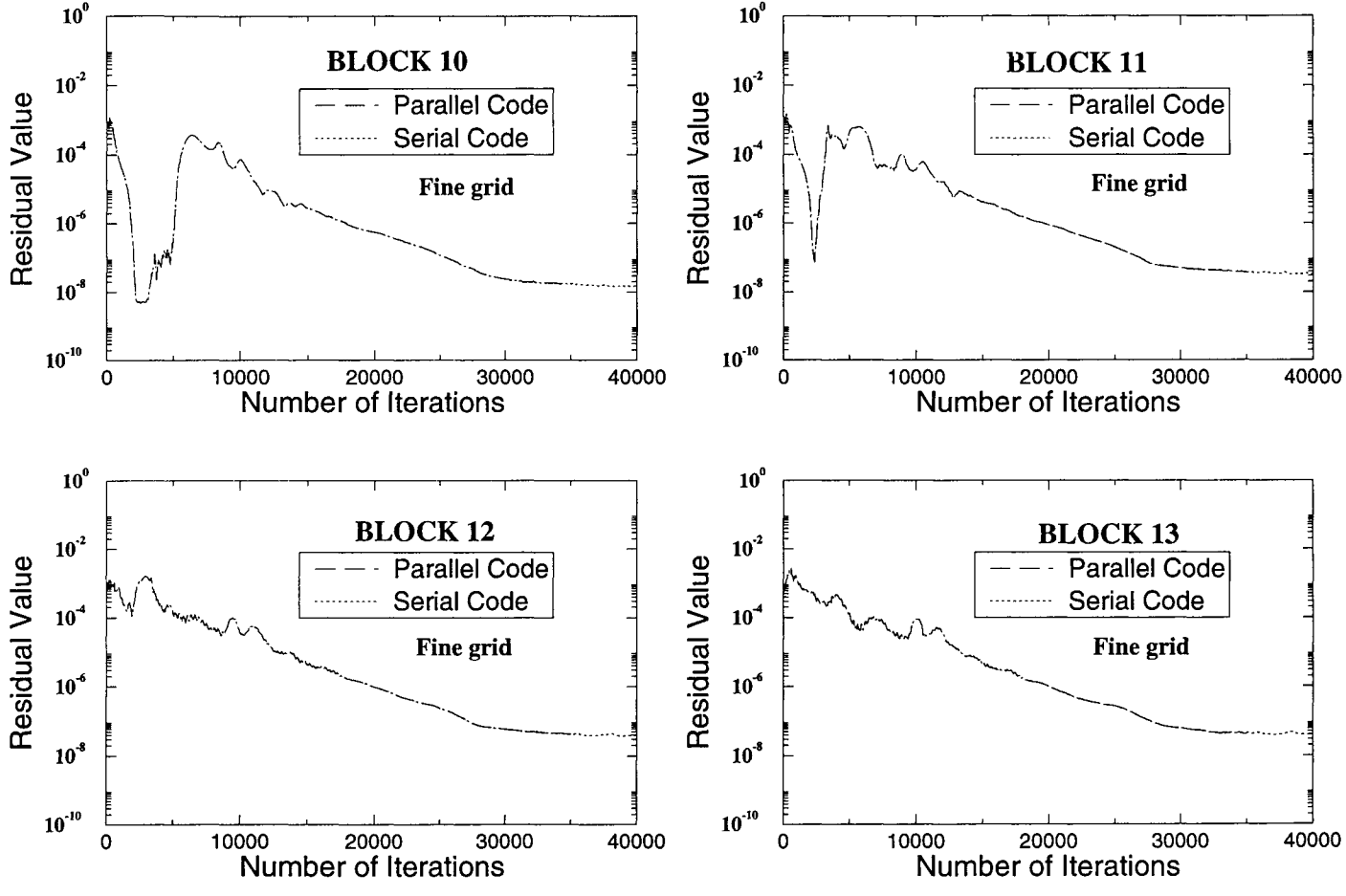


Figure 5. A plot of residual value versus number of iterations using parallel and serial codes

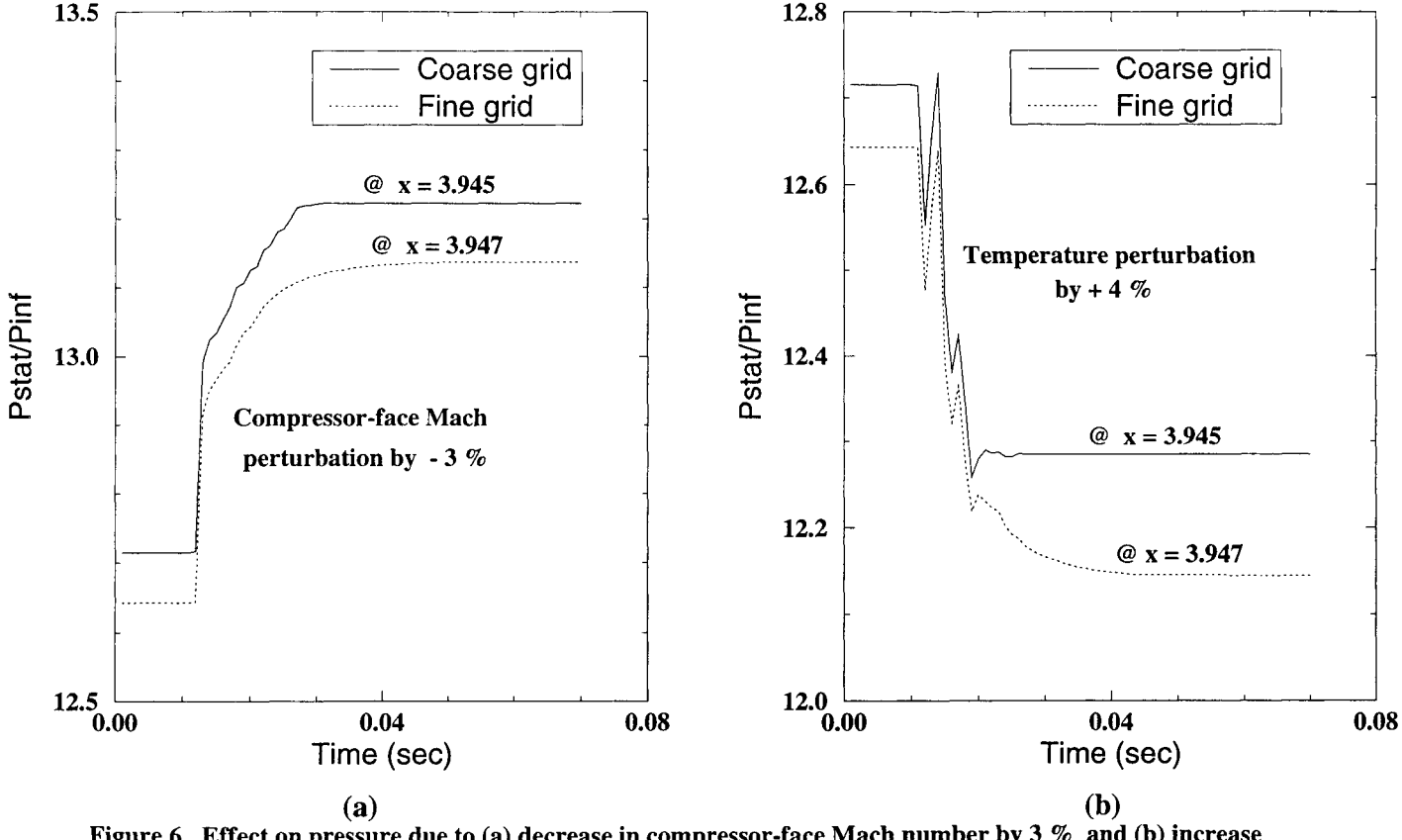


Figure 6. Effect on pressure due to (a) decrease in compressor-face Mach number by 3 % and (b) increase in free stream temperature by 4 %

This Page Intentionally Left Blank

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Parallel, Axisymmetric, Aerodynamic Simulation of a Jet Engine

Kim Ciula[a] and Mark E. M. Stewart[b]

[a]Sterling Software, 21000 Brookpark Road MS 142-5, Cleveland, OH 44135, USA
[b]NYMA, 2001 Aerospace Parkway, Brook Park, OH 44142, USA

1. ABSTRACT

A parallel, axisymmetric jet engine aerodynamic simulation is presented. The code solves the axisymmetric form of the fluid flow equations which include equation terms for the effects of blades, combustion and other aerodynamic components and effects. These equations are closed by reference to component simulations and solved for the engine system to capture steady component interactions. The code is demonstrated with a simulation of the Energy Efficient Engine. The cost advantage of clustered workstations and parallel computers over conventional supercomputers is forcing a shift in engineering and scientific computing which motivates this work. Further, a parallel computer is a natural environment for running and referencing the heterogeneous component codes which close the axisymmetric engine equations. This paper briefly discusses the full engine simulation and analyzes the code's performance, particularly near the communication bound performance limit experienced in workstation clusters.

2. INTRODUCTION

A jet engine can be characterized by a number of different components working together very efficiently at a range of demanding operating conditions. Several of these engine components are sensitive to interactions with neighboring components. For example, the efficiency of the compressor is very sensitive to steady inlet and outlet conditions, and a supersonic inlet is dependent on outlet pressure because fluctuations can unstart the inlet and expel the shock, substantially increasing drag and reducing engine performance. Consequently, during the design process it is important to consider not only isolated components but the engine as a system of components which influence each other.

Historically, the design process has started with a study of the complete proposed engine using performance maps and one-dimensional analysis. Then individual engine components are simulated and designed in detail by component design teams. Some engine components are designed by the airframe manufacturer. These results improve the performance maps and one-dimensional analysis, which helps address component interactions. These components are experimentally tested in isolation, progressively integrated, and adjusted to finalize the engine design.

Component design teams depend on analysis techniques to achieve the best performance. Streamline curvature methods (Smith,1966) continue to be extensively used to analyze multistage turbomachinery. Recently, the trend has been to apply advanced numerical techniques to engine components to understand the details of their operation in isolation. These applications range from quasi-three-dimensional blade calculations (Chima, 1986) which predict the behavior of a transonic blade to multistage compressor calculations (Adamczyk, 1985) which simulate the behavior of transonic compressors to simulation of nacelles and combustor chemistry.

These advanced analysis techniques do not account for engine component influences which are important when components are sensitive. The current work attempts to use advanced numerical techniques to capture these intercomponent influences through an engine system simulation.

It is believed in the jet engine analysis community that the geometrical and physical complexities of a jet engine are so great that it is necessary to depend on component analysis codes to complete an engine simulation. Using these codes takes advantage of their specialized expertise, careful verification and testing, the large investments in their development, and may help to avoid proprietary issues. In the current work, the results of these codes are used to close the axisymmetric formulation of the fluid flow equations for the full engine.

The current parallel implementation of the engine simulation was motivated by two factors. One is to examine the cost efficiency of using clustered workstations and parallel supercomputers. The other is to provide an environment where component analysis codes can execute and interact. Although the engine simulation currently references component code results using files, the eventual environment must involve communication with message passing.

To this end, the axisymmetric formulation of the equations is presented in the next section, followed by details of the numerical method, grid scheme, and the parallel implementation. Finally, solution and performance results are presented and discussed.

3. AXISYMMETRIC ENGINE FORMULATION

Engine aerodynamics can be described with an axisymmetric formulation. The equations may be derived from engine geometry and the three-dimensional fluid flow equations in a rotating cylindrical coordinate system by taking an average about the annulus (Jennions and Stow, 1985), (Stewart, 1995). In the relative frame, the equations relating the conservation of mass, momentum, and energy are

$$\frac{\partial}{\partial t}\int_{\Gamma} \overline{\mathbf{w}}\, b\, dV = -\oint_{\partial\Gamma} (\overline{\mathbf{F}}, \overline{\mathbf{G}})\cdot \mathbf{n}\, b\, dA - \int_{\Gamma} (\overline{\mathbf{k}}_{\Omega}\, b + \overline{\mathbf{k}}_{c} + \overline{\mathbf{k}}_{m}\, b + \overline{\mathbf{k}}_{r}\, b + \overline{\mathbf{k}}_{B} + \overline{\mathbf{k}}_{f}\, b + \overline{\mathbf{k}}_{bc}\, b)\, dV \tag{1a}$$

$$\overline{\mathbf{w}} = \begin{pmatrix} \overline{\rho} \\ \overline{\rho}\widetilde{W}_x \\ \overline{\rho}\widetilde{W}_r \\ \overline{\rho}\widetilde{W}_\theta \\ \overline{\rho}\widetilde{E} \end{pmatrix} \qquad \overline{\mathbf{F}} = \begin{pmatrix} \overline{\rho}\widetilde{W}_x \\ \overline{\rho}\widetilde{W}_x^2 + \overline{p} \\ \overline{\rho}\widetilde{W}_r\widetilde{W}_x \\ \overline{\rho}\widetilde{W}_\theta\widetilde{W}_x \\ \overline{p}\widetilde{I}\,\widetilde{W}_x \end{pmatrix} \qquad \overline{\mathbf{G}} = \begin{pmatrix} \overline{\rho}\widetilde{W}_r \\ \overline{\rho}\widetilde{W}_x\widetilde{W}_r \\ \overline{\rho}\widetilde{W}_x^2 + \overline{p} \\ \overline{\rho}\widetilde{W}_\theta\widetilde{W}_r \\ \overline{\rho}\widetilde{I}\,\widetilde{W}_r \end{pmatrix}$$

$$\overline{\mathbf{k}}_\Omega = \begin{pmatrix} 0 \\ 0 \\ -\frac{1}{r}(\overline{p} + \overline{\rho}(\widetilde{W}_\theta + \Omega r)^2) \\ \frac{\overline{\rho}\widetilde{W}_r}{r}(\widetilde{W}_\theta + 2\Omega r) \\ 0 \end{pmatrix} \quad \overline{\mathbf{k}}_c = \begin{pmatrix} \dot{\overline{\rho}}_f \\ 0 \\ 0 \\ 0 \\ -\overline{\rho}\dot{\tilde{q}} \end{pmatrix} \quad \overline{\mathbf{k}}_m = \begin{pmatrix} m_1(\overline{\rho}) \\ m_2(\overline{\rho}\widetilde{W}_x) \\ m_3(\overline{\rho}\widetilde{W}_r) \\ m_4(\overline{\rho}\widetilde{W}_\theta) \\ m_5(\overline{\rho}\widetilde{E}) \end{pmatrix} \quad \overline{\mathbf{k}}_\tau = \begin{pmatrix} 0 \\ -\widetilde{f}_{\tau_x} \\ -\widetilde{f}_{\tau_r} \\ -\widetilde{f}_{\tau_\theta} \\ 0 \end{pmatrix}$$

$$\overline{\mathbf{k}}_B = \begin{pmatrix} 0 \\ -\overline{p}\frac{\partial b}{\partial x} \\ -\overline{p}\frac{\partial b}{\partial r} \\ 0 \\ 0 \end{pmatrix} \quad \overline{\mathbf{k}}_f = \begin{pmatrix} 0 \\ -f_{b_x} \\ -f_{b_r} \\ -f_{b_\theta} \\ 0 \end{pmatrix} \quad \overline{\mathbf{k}}_{bc} = \begin{pmatrix} \dot{\rho}_{b_c} \\ \dot{\rho}_{bc}\widetilde{W}_x \\ \dot{\rho}_{bc}\widetilde{W}_r \\ \dot{\rho}_{bc}\widetilde{W}_\theta \\ \dot{\rho}_{bc}\widetilde{I} \end{pmatrix}$$

$$\overline{p} = Z(\overline{p}, \tilde{T})\,\overline{\rho}\,R_{gas}\,\tilde{T}(\tilde{e}). \qquad (1b)$$

There is also a torque equation for each spool of the engine. The overbar and tilde are used to denote circumferential and circumferential-mass averages, respectively. This circumferential averaging is not unlike the Reynolds average in time of the Navier-Stokes equation and also gives product of perturbation terms which are included in $\mathbf{f}_b$.

These equations (1) contain additional terms for combustion heating $\overline{\mathbf{k}}_c$, the effect of the mixer $\overline{\mathbf{k}}_m$, entropy increases $\overline{\mathbf{k}}_\tau$, blockage effects $\overline{\mathbf{k}}_B$, the isentropic components of blade forces $\overline{\mathbf{k}}_f$, and bleed and cooling $\overline{\mathbf{k}}_{bc}$. These terms are not closed or determined directly from these equations. Instead, the terms are calculated from the results of component simulations (Stewart,1995). In this way, the expertise of and investment in the codes which simulate diverse engine components may be used.

4. NUMERICAL METHOD

A steady-state approximation of these axisymmetric equations is found with an explicit, multi-stage Runge Kutta scheme (Jameson, 1981), where for each stage i

$$\overline{\mathbf{w}}^i = \overline{\mathbf{w}}^0 + \sigma_i \triangle t\,(\,Q(\,\overline{\mathbf{w}}^{i-1}\,) + D(\,\overline{\mathbf{w}}^{\min(i-1,1)}\,) + E(\,\overline{\mathbf{w}}^{i-1}\,)\,) \qquad i = 1:5. \qquad (2)$$

$Q(\overline{\mathbf{w}})$ is the boundary integral on the RHS of (1a), $E(\overline{\mathbf{w}})$ is the volume integral on the RHS of (1a), and $D(w)$ is an additional stabilizing dissipation.

The equations (1) are solved with this scheme in the engine's meridional plane. The complex geometry of this axisymmetric plane is discretized with a multiblock grid which covers the domain with topologically rectangular regional grids as shown in figure 1. These regional grids meet at interfaces without overlapping and coordinate lines are continuous across interfaces. The numerical treatment of the interfaces is to continue the solution up to two cell rows beyond the grid edge and copy solution variables from the adjacent block into these "halo cells". If solution data is passed from blocks to the halo cells of adjacent blocks at each stage of (2), then the interface is transparent to the explicit numerical scheme. The implementation of this scheme (2) on a multiblock grid is shown for a serial code in the left column of figure 2.

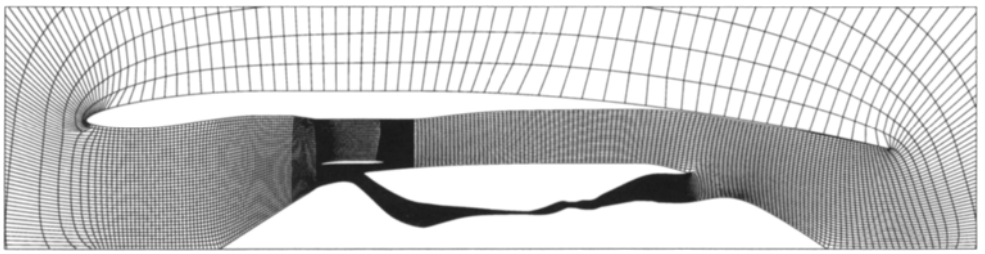


Figure 1: Inner region of the multiblock grid for the Energy Efficient Engine simulation. For clarity, only every other point is shown. The complete grid contains 114,080 cells.

Serial Code

```
    for each iteration
       for each block
[2.8]     calculate △t
       endfor–block
       update halo cells for △t
       for each stage i = 1 : 5
          for each block
[20.]        if i < 3 calculate D(w̄)
[30.]        calculate Q(w̄)
[15.]        calculate E(w̄)
[13.]        w̄^i = w̄^0 + σ_i △t(Q + D + E)
[10.]        wall boundary conditions
             update halo cells for w̄^i
          endfor–block
       endfor–stage

[3.]   calculate other boundary conditions
       update halo cells for w̄^5
[1.4]  evaluate thrust, torque, convergence
    endfor–iteration
```

Parallel Code on Each PE

```
for each block on PE
   calculate △t
   send △t to halo cells
endfor–block
for each iteration
   for each stage i = 1 : 5
      for each block on PE
         if i = 1 receive halo △t
         receive halo w̄^(i-1)
         if i < 3 calculate D(w̄)
         calculate Q(w̄)
         calculate E(w̄)
         w̄^i = w̄^0 + σ_i △t(Q + D + E)
         send w̄^i to halo cells
         wall boundary conditions
      endfor–block
   endfor–stage
   for each block on PE
      calculate △t
      send △t to halo cells
   endfor–block
   calculate other boundary conditions

   evaluate thrust, torque, convergence
endfor–iteration
```

Figure 2: Optimized serial and parallel codes for the numerically intensive part of the simulation; the initialization and postprocessing are excluded. The percentage of serial program CPU time spent in each routine is given in brackets. A copy of the parallel code resides on each PE, and messages correspond to sending halo cells. Italics highlight code differences.

5. PARALLEL IMPLEMENTATION

The multiblock grid was originally used to discretize the complex geometry; however, this decomposition strategy facilitates the parallel implementation. The grid blocks are distributed among the processing elements (PEs), possibly with multiple blocks per PE, to give a coarse-grained parallelism. The halo cells at block interfaces are updated with adjacent block solution data in PVM 3.3 (Geist, 1993) messages when the adjacent blocks span PEs or with memory copies for adjacent blocks on a single PE. If the halo cells are updated at each stage of (2), the interface is again transparent to the numerical scheme.

This parallel numerical scheme is shown in the right column of figure 2. To ensure that identical results are produced by the serial and parallel codes, a suite of six test cases were run and the serial and parallel results were compared in terms of convergence quantities and solution results. The results were identical.

6. ANALYSIS AND RESULTS

To estimate the performance of the numerically intensive part of the parallel code (figure 2), a test case and performance model were devised. This testcase is an annular duct which is subdivided along one axis, and the model is

$$t_{\text{wall}} = t_{\text{computations}} + t_{\text{communications}} = \frac{Cv}{N} + 6d. \tag{3}$$

Here t is the time for one iteration, C is the number of cells in the problem, v is the program speed per cell per iteration, N is the number of PEs, and d is the average message time for each of the six messages passed in each iteration. A further limit on performance is that the required bandwidth must not exceed the measured network bandwidth. For shared networks (Ethernet) and point-to-point networks the required bandwidth is estimated as

$$B_{\text{shared}} = \frac{\alpha\, bN}{d} \text{ MB/s} \qquad B_{\text{point-to-point}} = \frac{\alpha\, b}{d} \text{ MB/s.} \tag{4a,b}$$

Here B is the bandwidth, b is the number of cells of data which is sent in each message (here $b = 128$), and α is a conversion constant from cells to megabytes ($\alpha = 4.8\text{x}10^{-5}$).

The parallel code development environment consisted of sixteen IBM RS/6000 model 590 workstations, each with at least 128 MB of memory. These PEs were networked with both an Ethernet shared network ($B_{\text{ideal}} = 1.25$MB/s, $B_{\text{measured}} = 1.$MB/s) and an Asynchronous Transfer Mode (ATM) switch ($B_{\text{ideal}} = 12.5$MB/s, $B_{\text{measured}} = 5.$MB/s) from FORE systems. Benchmark tests were conducted on a dedicated cluster and indicated that $d = 9.\text{x}10^{-3}$ seconds (ATM) and $v = 5.\text{x}10^{-5}$ seconds/cell/iteration on each PE. For comparison, on a single processor Cray YMP 8E/8128, serial code tests indicate that $v = 9.\text{x}10^{-6}$ seconds/cell/iteration.

Comparison of benchmark tests and the model (3, 4) indicate that the model is optimistic about speedup. Even with more detailed models than (3, 4), the communications cost is significant at four times fewer PEs than predicted. To reduce communications cost, several techniques are used to overlap communications and computations and reduce delays due to communications. First, blocks were assigned to processors such that

one block's communications could overlap the computations and communications of the other blocks on that PE. Code tests indicate that overlapping blocks doubles performance on 8 PEs as shown in figure 3. Second, code segments were reordered to further overlap communications and computations, further improving performance (figure 3).

Figure 4 shows how the latter three stages of the numerical method require fewer computations than the first two, and consequently require a shorter message time-of-flight to avoid delays. The code was modified so messages are not sent or received in stage four. With this treatment, the interfaces are no longer transparent to the numerical scheme. However, this conservative change improves performance 20% on 8 PEs as shown in figure 3.

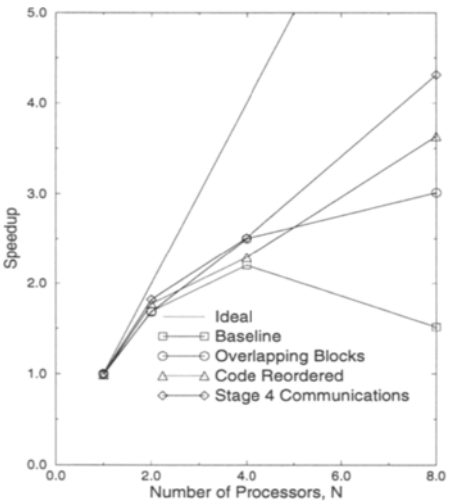


Figure 3: Speedup for a 32,768 cell annular duct grid for cumulative code optimizations.

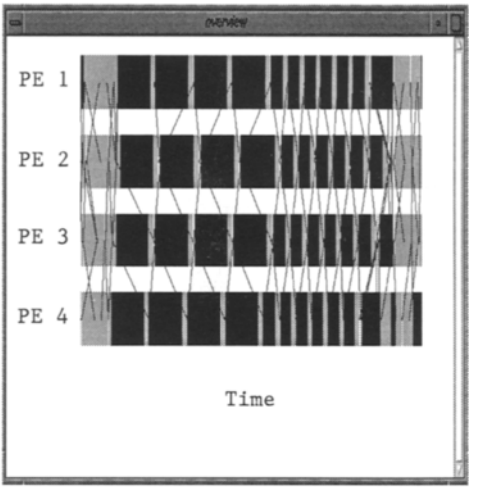


Figure 4: AIMS processor and message activity graph for the 32,768 cell annular duct grid with eight blocks on four PEs.

To emphasize the sensitivity of the code to the computation to communication ratio, the optimized code was run with a higher resolution annular duct grid (262,144 cells). Speedup results are shown in figure 5.

The experience of this testcase and optimizations was applied to a simulation of the Energy Efficient Engine (EEE) which was designed and tested by General Electric for NASA in the 1980's. It is representative of modern commercial jet engines and is a forerunner of the GE90 high-bypass turbofan engine. The engine geometry and computational grid (144,080 cells) are shown in figure 1. Figure 7 shows the computed Mach contours at the cruise design point (M=0.8, 35,000 ft). The grid was blocked to contain 40 blocks which were distributed on the PEs to give a load balance, $\frac{\text{avg PE cells}}{\text{max PE cells}}$ of 0.96 to 0.90 on 8 PEs. Figure 6 shows the speedup for the Energy Efficient Engine simulation.

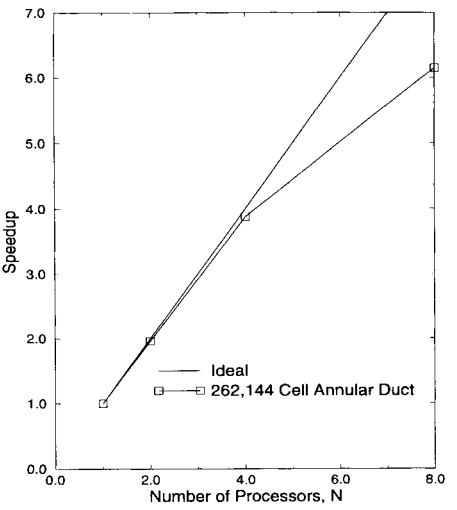


Figure 5: Speedup for a 262,144 cell annular duct grid.

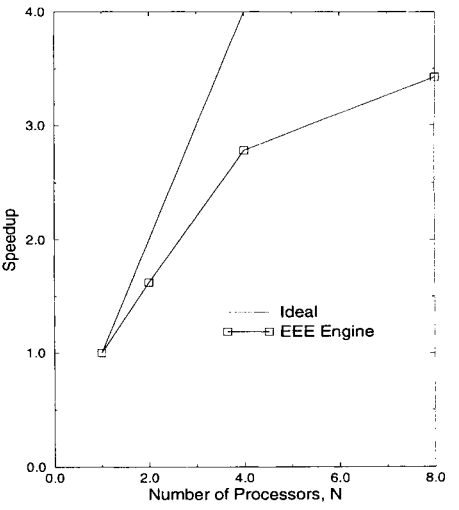


Figure 6: Speedup for the 114,080 cell grid of the Energy Efficient Engine simulation.

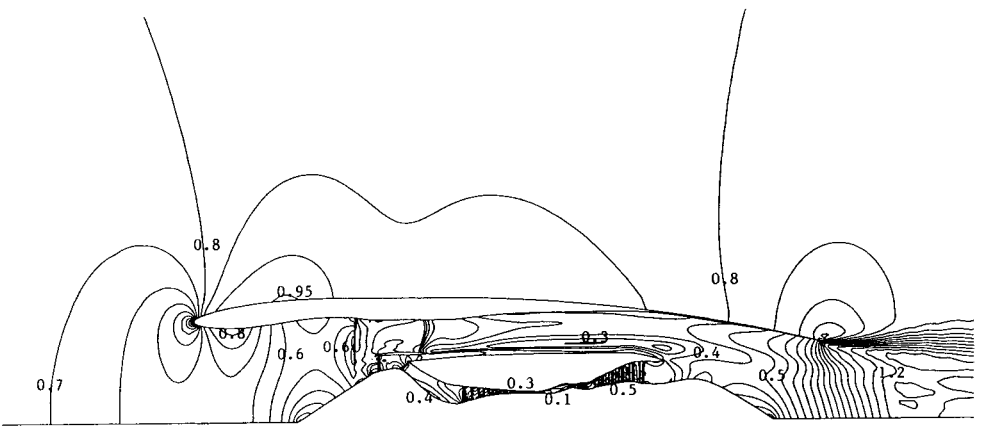


Figure 7: Computed Mach contours for the Energy Efficient Engine at the cruise design point: Mach 0.8, 35,000 ft.

7. CONCLUSIONS

A system for full engine simulation has been developed and parallelized on a workstation cluster using PVM. The relatively small grids involved and the demands that the numerical algorithm place on the network result in a communication-bound problem, limiting speedup. Testing with the faster networks of the Cray T3D and IBM SP2 is in progress. However, economic necessities have spurred the move to relatively inexpensive workstation clusters. The code is now being directly integrated with component codes to create a system of heterogeneous intercommunicating codes to improve the process of engine design and analysis.

8. ACKNOWLEDGEMENTS

This work was completed at the NASA Lewis Research Center and was supported by the Computational Sciences group under contract NAS3-27121 with Jay Horowitz as monitor and by the Numerical Propulsion System Simulation project under contract NAS3-27186 with Austin Evans and Russell Claus as monitors. Scott Townsend has generously shared his experience with parallel computing. Udayan Gumaste wrote a heuristic load balancing algorithm for the EEE engine grid. The authors would also like to thank Ehtesham Hayder, Angela Quealy, Melisa Schmidt, and Jerry Yan for their assistance in this work.

REFERENCES

Adamczyk, J. J., 1985, "Model Equation for Simulating Flows in Multistage Turbomachinery," ASME Paper 85-GT-226.

Chima, R. V., 1986, "Development of an Explicit Multigrid Algorithm for Quasi-Three-Dimensional Viscous Flows in Turbomachinery," NASA TM-87128.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V., 1993, "PVM 3 User's Guide and Reference Manual," Oak Ridge National Laboratory.

Jameson, A., Schmidt, W., Turkel, E., 1981, "Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time Stepping Schemes," AIAA Paper 81-1259, pp. 327-356.

Jennions, I. K., Stow, P., 1985, "A Quasi-Three-Dimensional Turbomachinery Blade Design System: Part 1 - Throughflow Analysis," ASME Journal of Engineering for Gas Turbines and Power, Vol. 107, pp. 301-307.

Smith, L. H., 1966, "The Radial-Equilibrium Equation of Turbomachinery," ASME Journal of Engineering for Power, pp. 1-12.

Stewart, M. E. M., 1995, "Axisymmetric Aerodynamic Numerical Analysis of a Turbofan Engine," ASME Paper 95-GT-338.

Yan, J., Hontalas, P., Listgarten, S., "The Automated Instrumentation and Monitoring System (AIMS) Reference Manual," NASA TM-108795.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

An investigation of load balancing strategies for CFD applications on parallel computers

N. Gopalaswamy[a], Y.P. Chien[a], A. Ecer[a], H.U. Akay[a], R.A. Blech[b] and G.L. Cole[b]

[a]Purdue School of Engineering and Technology, IUPUI, Indianapolis, Indiana

[b]Computational Technologies Branch, NASA Lewis Research Center, Cleveland, Ohio

1. INTRODUCTION

As the use of parallel computers is becoming more popular, more attention is given to manage such systems more efficiently. In this paper, several issues related to the problem of load balancing for the solution of parallel CFD problems are discussed. The load balancing problem is stated in a general fashion for a network of heterogeneous, multi-user computers without defining a specific system. The CFD problem is defined in a multi-block fashion where each of the data blocks can be of a different size and the blocks are connected to each other in any arbitrary form. A process is attached to each block where different algorithms may be employed for different blocks. These blocks may be marching in time at different speeds and communicating with each other at different instances. When the problem is defined in such general terms, the need for dynamic load balancing becomes apparent. Especially, if the CFD problem is a large one, to be solved on many processors over a period of many hours, the load balancing can aid to solve some of the following problems:

- load of each processor of a system can change dynamically on a multi-user system; one would like to use all the processors on the system whenever available.
- an unbalanced load distribution may cause the calculations for certain blocks to take much longer than others, since the slowest block decides the elapsed time for the entire problem. This may occur during different instances of the execution if the algorithm is dynamic, i.e., solution parameters change with the solution.

Based on the above considerations, the load balancing problem was treated by dynamically adjusting the distribution of the blocks among available processors during the program execution, based on the loading of the system. The details of the load balancing algorithm was presented previously [1,2]. Here, only the basic steps of the dynamic load balancing process are listed as follows:

- Obtain reliable computational cost information during the execution of the code.
- Obtain reliable communication cost information during the execution of the code.
- Determine the total cost in terms of computation and communication costs of the existing block distribution on the given system.
- Periodically, re-distribute the blocks to processors to achieve load balancing by optimizing the cost function.

In the present paper, an Euler and Navier-Stokes code, PARC2D, is employed to demonstrate the basic concepts of dynamic load balancing. This code solves unsteady flow equations using conservation variables and provides different order Runge-Kutta time-stepping schemes [3]. Parallel implementation of the explicit time-integration algorithm involves the following steps:

- Division of the computational grid into a greater number of blocks than the number of available processors with one layer of elements overlapped at inter-block boundaries.
- Introduction of the block and interface information into the data base management program, GPAR [4].
- Distribution of the blocks and associating interfaces among the available processors by using GPAR.
- Definition of PARC2D as a block-solver for the solution of the equations inside each block either for Euler or Navier-Stokes equations.
- Preparation of an interface solver to communicate with its parent block and its twin interface. As can be seen from Figure 1, each block automatically sends information to its interfaces after each iteration step. The interfaces will then send information to their twins whenever necessary for the twin to update its parent block. The task assigned to each block may not be identical, due to factors such as: size of the block, choice of either Euler vs. Navier-Stokes equations for a particular block, size of the time-step for solving each block and the time-step for communicating between the interfaces. Thus, controlling communications and computations in such a truly heterogeneous environment becomes even more critical.

These issues are discussed below in detail, for a sample problem.

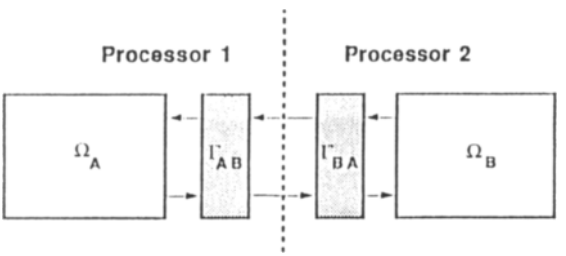


Figure 1. Communication between two neighboring blocks and related interfaces (Ω_A and Ω_B are blocks, Γ_{AB} and Γ_{BA} are interfaces).

2. INVESTIGATION OF DYNAMIC LOAD BALANCING STRATEGIES

Numerical experiments were chosen to demonstrate some strategies in dynamic load balancing for managing computer systems and algorithms. The chosen test problem is a two-dimensional grid for an inlet with 161,600 grid points as shown in Figure 2. The flow region is divided into 17 blocks as shown in Figure 3, each with approximately 10,000 grid points.

Figure 2. Computational Grid for the Inlet (161,600 nodes).

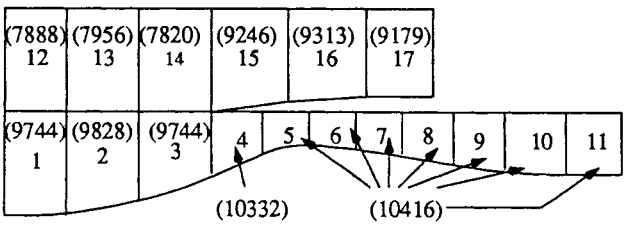


Figure 3. Division of the flow-field into Blocks for the Inlet Grid (number of grid points in each block are shown in parentheses).

2.1. Load Balancing for Variations in the System Loading

One type of load balancing strategy involves controlling the computer system. It may be a heterogeneous and multi-user system. It is also dynamic in a sense that it changes over a long run. The test problem was run on four processors over a period of approximately twelve hours. Communication and computation costs for each process were recorded and a load balancing was performed after approximately every thirty minutes. Figure 4 summarizes the results of this computation for a controlled environment. As shown in Figure 4a, the loading of certain machines was increased by adding extraneous processes, while on other machines no other jobs are running. The response of the load balancer is summarized in Figure 4b. Over 24 load balance cycles, the elapsed time for each iteration varies between 1.5 to 4 seconds. The load balancer recorded the communication and computation cost data over a cycle and predicted the elapsed time for a suggested load balanced distribution. As can be seen from this figure, the prediction is quite accurate and reduced the elapsed time by removing the bottlenecks. Figure 5 illustrates the same problem run on an uncontrolled environment. Four heavily used processors were chosen during the daytime operation. The load balancer responded in a similar fashion to a rather irregular loading pattern of the system. It is interesting to note that in

this case, the total elapsed time was not excessive in comparison with the elapsed time for the dedicated machine.

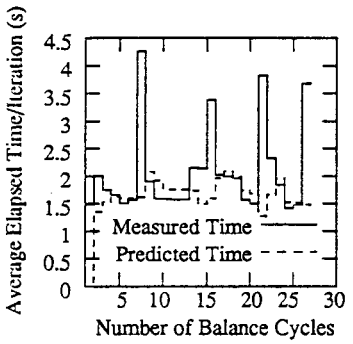


Figure 4a. Load balancing in a controlled environment.

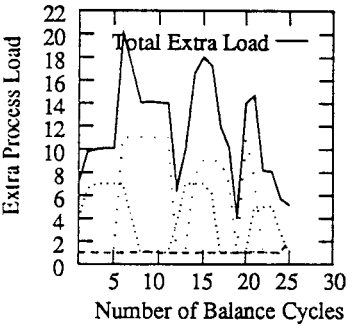


Figure 4b. Extra load variation on the system for the controlled environment.

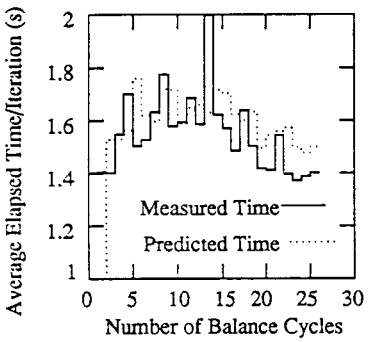


Figure 5a. Load balancing in a multi-user environment.

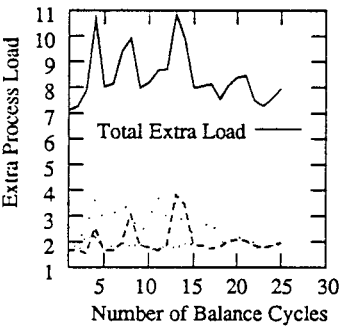


Figure 5b. Extra load variation on the system for the multi-user environment.

2.2. Load Balancing for Heterogeneous Algorithms in Parallel Computing

The second type of load balancing strategy involves optimizing the algorithm on a parallel system and dynamically load balancing the problem as the algorithm adapts to the solution. When running the PARC2D code, one can specify a time step for each block from the CFL condition as defined below:

$$\Delta t = \mathrm{CFL} \;/\; \mathrm{Max}_{j}\left[\left(\left|U_j\right| + a\left|K_i^j\right|\right) + \frac{2}{\mathrm{Re}}\frac{\mu}{\rho}\left|K_i^j\right|^2\right] \tag{1}$$

where U_j are the contravariant velocities, a is the speed of sound, Re is the Reynolds number, μ is the viscosity, ρ is the density, and K_i^j is the Jacobian matrix. This time

step is calculated for all the grid points inside a block, depending on the local flow conditions and grid size; the minimum value of all such time steps is chosen as the time step for that particular block. Since, the flow conditions are changing, the time step for each block changes over the history of a complete run [5].

Variable time-stepping with variable communications is illustrated in Figure 6a for two neighboring blocks on two different processors. In this case, Δt_{min} is a global reference time step for all the blocks. The first block at this instant is operating with a time step of $3\Delta t_{min}$, while the second block is running with $2\Delta t_{min}$. The arrows indicate the instances at which an interface of a block sends a message to its neighbor. Figure 6b shows a non-optimum solution. Here, while the computations are performed for each block solver with its own time step, each block is sending information to its neighbor at every global time step.

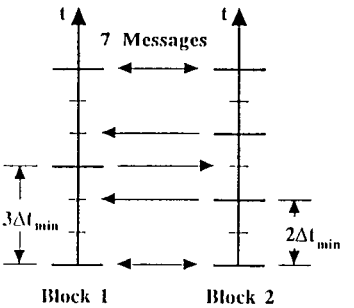


Figure 6a. Communication occurs when necessary

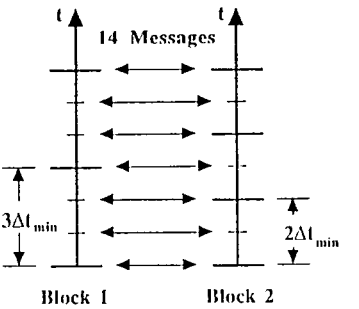


Figure 6b. Communication occurs at the global time step.

Figure 7 provides a summary of the computations with fixed and variable time-stepping. The reference case is case 1, where the time step is the same for all the blocks.

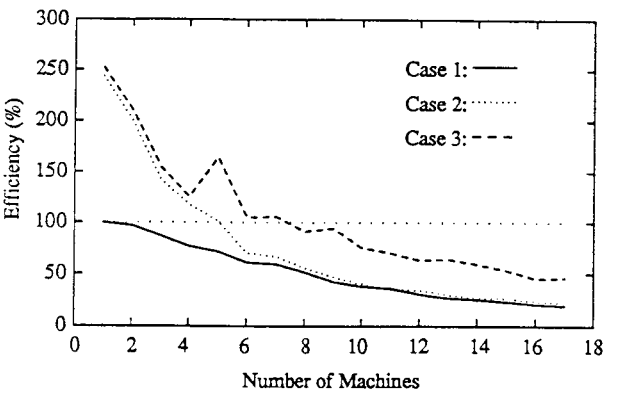


Figure 7. Parallel efficiency vs. number of machines.

The equations are solved for each block at each time step and the blocks communicate with each other after each time step. As can be seen from this figure, the parallel efficiency falls below 50% after 8 processors for the sample problem. Case 2 indicates the importance of variable time-stepping that is local to each block. In this case, each block chooses its own time step for solving the equations for that block, however communicates with its neighbors based on the global time step, as described in Figure 6b. As can be seen from Figure 7, after 6 processors, communication cost becomes the dominant factor for this case. Case 3 illustrates the need for intelligent communication as suggested by Figure 6a. In this case, a block sends a message to its neighbor only when necessary since each block is solved only when necessary. In this case, the parallel efficiency can be maintained at a higher level even when the communication cost becomes dominant. For example, around the leading edge of an airfoil with very fine grids, one can choose time steps of different order than other blocks and save computation time. Also, these blocks may not need to talk to their neighbors after each solution time step. The computational savings, discussed above, are purely due to the refinements in the use of the algorithm. When performing parallel computing, one can localize the algorithm according to the flow conditions and grids, especially for the solution of large problems with complex grids. It should be remembered that all of the above cases were load balanced to determine the most efficient distribution under given conditions. These experiments were possible only after a reliable load balancing procedure was developed.

The second example involves the solution of Euler and Navier-Stokes solutions at different blocks. The time step restriction for viscous computations is more restrictive than Euler computations as can be observed from Equation 1. Figure 8 illustrates a case when the computations were started by an Euler computation for blocks 12-17 and Navier-Stokes solution for blocks 1-11.

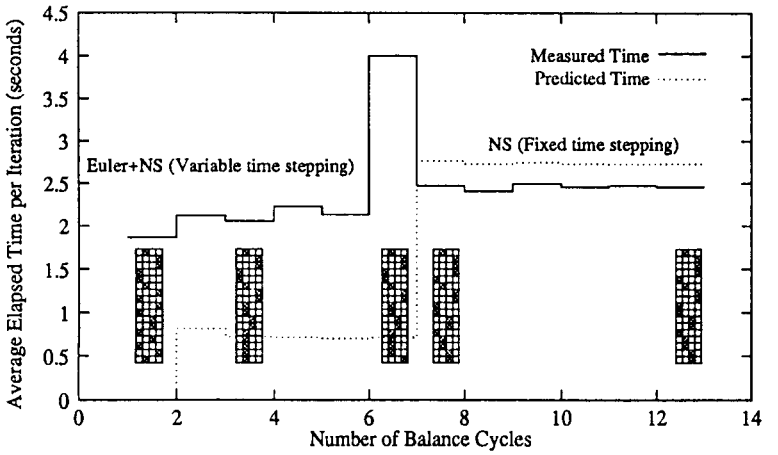


Figure 8. Load balancing due to change in solution algorithm.

The numerical integration took approximately 2.1 seconds per time-step. Local time-stepping was employed for all blocks. The distribution of the 17 blocks among 4 machines is also shown in the figure. Afterwards, blocks 12-17 were switched to a Navier-Stokes solver and global fixed time-stepping was employed for all blocks. As can be seen again from this figure, the load balancer provided a new distribution which eliminated the bottleneck by removing several processes from machine 2 and loading machines 1 and 3. Again in this case, it is shown that an algorithm can be defined and executed locally on a flow region for improving efficiency. By defining the parallel computing in a heterogeneous environment, one can employ an algorithm in a most efficient manner whenever necessary.

The third example relates to the development of algorithms which communicate in a selective manner. The cost of communication is still the dominant factor in parallel computing. It only makes sense to develop intelligent interfaces to communicate between the blocks-processes. Figure 9 shows two blocks in a one-dimensional flow field which are sending messages to each other at different speeds.

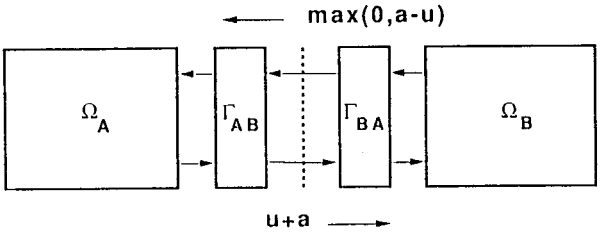


Figure 9. Communication in subsonic and supersonic flows.

Also by remembering the grid requirements for the grid points on an interface, send and receives between the two neighboring blocks can be executed at different time intervals. The test case is a specific one where most of the flow is supersonic except for blocks 8-11 which are located inside the inlet. In this case for all supersonic interfaces, one can send messages only in one direction. Figure 10 demonstrates such a case. The time-integration started where each block was communicating with its neighbors as discussed above. The distribution of the blocks among the processors is also shown in the figure. The solution scheme was then modified where the supersonic flow regions the messages were sent only in one direction. The load distribution was also modified as shown in this figure which reduced the elapsed time per iteration from 2.5 to 1.8 seconds. This figure also shows a change in the loading of the system after the 13th balance cycle which was corrected by the load balance: a block was moved from machine 3 to machine 4.

The above examples illustrate the advantages of parallel computing defined in a general fashion. Concepts such as heterogeneity and asynchronous computations in terms of both algorithms and computer systems can help to improve efficiency of parallel computing.

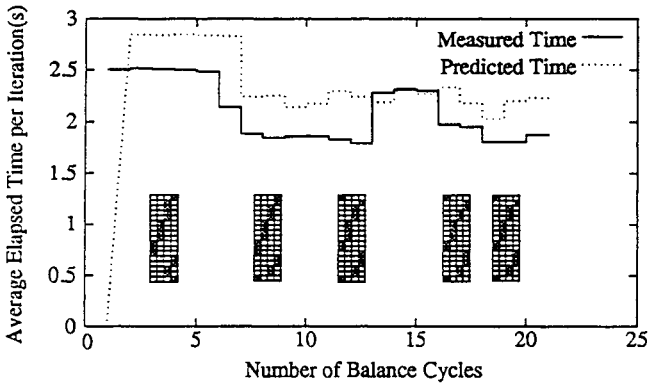


Figure 10. Load balancing in subsonic and supersonic flows.

ACKNOWLEDGMENTS

This research was supported by the NASA Lewis Research Center under NAG3-1577. Computer access provided by NASA and IBM is gratefully acknowledged. The authors thank S. Secer of IUPUI's CFD Laboratory for his assistance on the implementation of the load balancing algorithm.

REFERENCES

1. Y.P. Chien, A. Ecer, H.U. Akay, F. Carpenter and R.A. Blech, "Dynamic Load Balancing on a Network of Workstations for Solving Computational Fluid Dynamics Problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 119, pp. 17-33, 1994.
2. Y.P. Chien, A. Ecer, H.U. Akay and R.A. Blech, "Environment Requirements for Using Dynamic Load Balancing in Parallel Computations," *Proceedings of Parallel CFD '94*, Edited by A. Ecer et al., Elsevier, Amsterdam, 1995.
3. G.K. Cooper and J.R. Sirbaugh, "The PARC Code: Theory and Usage," Arnold Engineering Development Center, TR-89-15, 1989.
4. H.U. Akay, R.A. Blech, A. Ecer, D. Ercoskun, B. Kemle, A. Quealy and A. Williams, "A Database Management System for Parallel Processing of CFD Algorithms," *Parallel CFD '92*, Edited by R.B. Pelz, et al., Elsevier, Amsterdam, pp. 9-23, 1993.
5. H.U. Akay and A. Ecer, "Efficiency Considerations for Explicit CFD Solvers on Parallel Computers," *Proceedings of the International Workshop on Solution Techniques for Large-Scale CFD Problems*, Montreal, Quebec, Canada, pp. 289 - 314, September 26-28,1994.

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

CPULib — A Software Library for Parallel Applications on Arbitrary Meshes *

Michael Rudgyard† and Thilo Schönfeld [a]

[a]CERFACS, 42 av. Gustave Coriolis, 31057 Toulouse, France.

We present work on the development of the CERFACS Parallel Utilities library. This prototype software tool aims to simplify the task of parallelising large 2d and 3d calculations on grids of arbitrary elements using a variety of numerical algorithms. Although simple to use, it is designed to be highly efficient.

1. Introduction

Although parallel computing has become a subject in its own right, most users of high performance computing do not wish to become involved in the details of parallelisation. Parallel libraries are therefore becoming increasingly popular. Notable examples in the field of numerical simulation are PARTI [2] and OPLUS [1]. Such libraries free the non–specialist user from the need to consider data partitioning, message passing, parallel I/O and the data–structures that are required to implement these. Because the user interacts with the library through simple subroutine calls, it is possible to optimise inter–processor communications as well as computationally intensive parts of the library independently. This increases the portability of the resulting application code whilst retaining efficiency on particular hardware platforms.

The **C**ERFACS **P**arallel **U**tilities library, or **CPULib**, is similar in concept to OPLUS. Both are written in FORTRAN 77, are applicable to arbitrary grids in two and three dimensions, use a *data parallel* strategy, and include integrated partitioning algorithms. They also impose similar restrictions on the types of algorithms that may be implemented within the context of the library. In general, CPULib is aimed at a slightly more experienced user. Unlike OPLUS, which uses a single parallel strategy of *owned* and *unowned* sets, CPULib permits the user to choose between various *accumulation* and *data copy* algorithms, with *owned* and *unowned* variants; arbitrary overlapping of interface sets is also permitted. This is included at the expense of some minor additional complexity, since the user must be aware of the rudiments of parallel computing. However, he is not required to deal with message–passing at a low level, nor does he need to store or calculate any of his own pointers for treating inter–partition communications.

CPULib is built upon the IPM macros developed at CERFACS [3]. These enable us to choose between the message–passing libraries M4, PVM, PARMACS and MPI (as

*Work partly funded within the EC HCM programme, Contract No. CHRXCT920042
†Presently: Smith Research Fellow, Oxford University Computing Laboratory, The Wolfson Building, Parks Road, Oxford OX1 3QD, England

well as manufacturers' optimised libraries) by using the standard preprocessor M4. The software has been ported to a range of parallel machines including workstation clusters, the MEIKO CS2, the IBM SP1 and SP2, the CRAY T3D, and the INTEL PARAGON.

2. Implementation Overview

The prototype version of CPULib is based on a master–slave or client–server implementation. The user writes the code that corresponds to the slave processes and communicates with a generic master process, as well as other slaves, using subroutines from the library. The master process first spawns copies of the application code, which in turn starts the library session, thereby initialising all relevant data structures. The user must then provide CPULib information that enables grid connectivity pointers to be defined, as well as the pre–defined sets over which data is to be stored. On receiving this information, the master process reads a set of user–specified grid files in a standard format, creates the required pointers, and then partitions these along with the grid coordinates; the relevant information is then made available to the individual slave applications.

Once this initialisation phase has been completed, the master enters a control loop and awaits any further messages that contain specific directives from the slaves — these arise when the user wishes to input or output data, end the CPULib session, or abort the parallel application. Simple routines for inter–processor communications allow the user to update the elements of distributed arrays that correspond to data stored on or near partition interfaces. Other routines exist for calculating global operations on these arrays, whilst basic support for some standard iterative methods is also provided. An example of a typical application programme is given in Fig 1.

Although the master–slave paradigm is suitable for several popular parallel computers, an SPMD implementation is arguably more appropriate for many newer machines that support batch–queueing and single–user modes. Whilst this is not available at present, the standard subroutine interface allows us the freedom to implement an SPMD version of CPULib in a simple manner (eg. by allowing the root node to effectively act as both the master and a slave). This is also true if a truly scalar code is required. In either case, the user would not have to change his application programme at all.

3. Partitioning and Interface Treatments

Several partitioning techniques are available within the library and once the appropriate choice has been made by the user, CPULib begins by subdividing the mesh into non–overlapping subsets of *elements*, suitably load–balanced. Such an approach is compatible with the concept of multi–block structured meshes (the library is able to deal with such meshes, although it treats them as unstructured sets of hexahedra within each block; however, support for fully structured blocks is planned). By definition, distributed data over nodes, edges or faces includes copies of interface values in all partitions adjacent to the interface. However, these copies are distinguished as being *owned* or *unowned* by appropriate processors so that all sets are effectively partitioned; a simple heuristic is used to load–balance these. The subsets that lie on the original interface are thus distinguished as *shared/owned* and *shared/unowned.* CPULib takes this distinction one step further in its internal representation, since a *communication patch* between two given processors

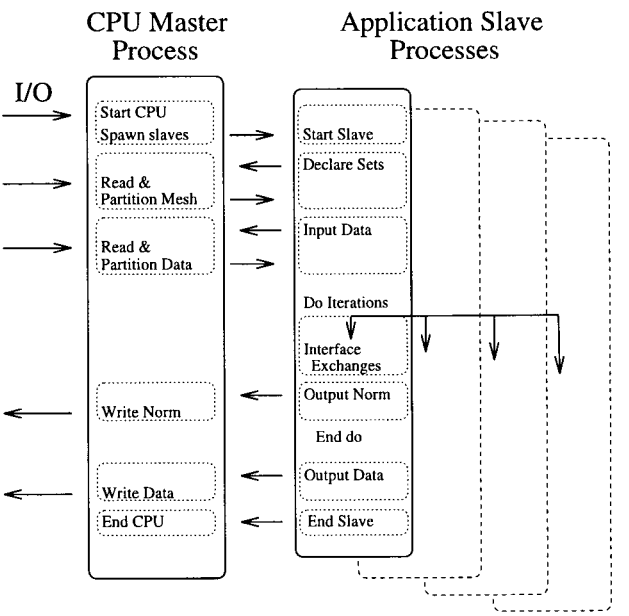


Figure 1. A typical Master–Slave Application using CPULib

differentiates between subsets that are owned by the present processor, owned by the neighbour or owned by neither — this enables the library to update owned or unowned values on neighbouring processors without any unnecessary data being exchanged.

CPULib also supports two types of overlapping partitions. The first, *partial overlapping*, defines new layers of elements around owned interface values. Copies of distributed arrays may then be stored here, as well as at the new nodes, edges or faces that have been created. If *full overlapping* is required, new layers of elements are built around the original partition interface, and not only owned subsets. In both cases, CPULib automatically sets up the communication pointers that are required to update any new values within distributed arrays. The situation is represented graphically in Figure 2.

It is useful to describe the need for these different interface treatments by considering a specific example. Let as assume that during the course of an iteration within a typically CFD application, the user wishes to construct residual values at nodes by using a double loop over elements — the first loop could correspond to the construction of artificial viscosity terms, or nodal gradients that are to be used in a MUSCL–type scheme; the second loop may then complete the construction of the residual. If the data itself is stored over nodes, indirect addressing via the element–to–node pointer is required, and each loop uses appropriate gather and scatter operations to accumulate the nodal values.

Within the framework of CPULib, several different parallel algorithms may be used to implement this double loop, and only some of these will be discussed here. If the residual

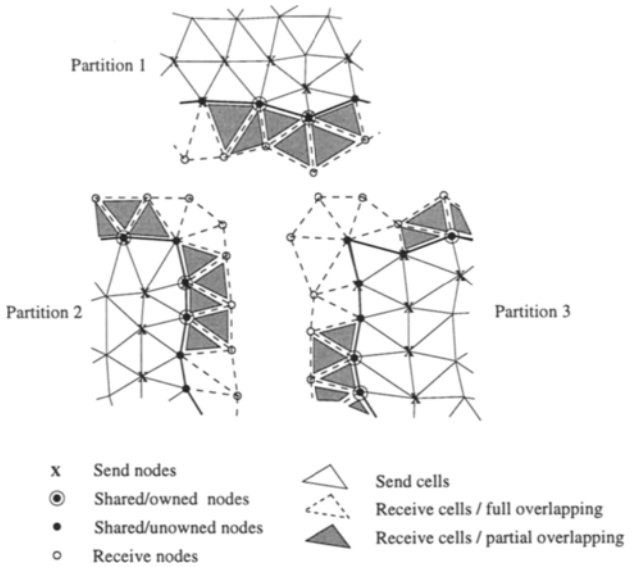


Figure 2. Partition Interfaces with overlapping

is only required at owned nodes within each partition, the user may choose to employ one layer of partial overlapping and a data–copy algorithm. In this case, data at all unowned nodes is copied before executing a loop over both owned and unowned elements; at the end of each scatter phase, the required subset of nodal values is then correct. Alternatively, and if no overlapping is desired, values may be accumulated at the complete shared interface by calling the appropriate CPULib routines *following* each of the cell loops; this results in up–to–date values at both owned and unowned nodes (a simple owned–node variant is also possible, although this requires two communication phases for each loop). A technique that uses a single communication phase for the complete double loop is also possible. In this case full overlapping is chosen — the first loop considers both owned and unowned elements and results in correct nodal values for the original non–overlapping partition; following the second loop that is executed over owned elements only, the result is accumulated at all nodes using an inter–processor exchange.

The choice as to which of these algorithms is suitable for a given numerical scheme and/or a given hardware platform may be complicated. If most of the computations are based on elements, and the inter–processor communications are reasonably efficient, then the non–overlapping accumulation algorithm is likely to be preferable since it avoids any redundant element calculations. If message–passing is slow due to high latencies, then a single communications phase may be optimal, and the fully–overlapping accumulation algorithm may work best. However, if a complex iterative method is used (where the nodal operations may be much more expensive than those based on elements), and the communications bandwidth of the machine is high, then the data–copy algorithm may

prove to be the most valuable. Note that other details of the algorithm may also be important: eg. if the first of the cell loops described above serves to calculate a solution gradient, then it may be wise to avoid exchanging these values at interfaces rather than the solution itself (which will have fewer components, and hence messages will be shorter...).

4. Specific Functionality

4.1. Initialisation and Control Routines

Several CPULib initialisation and control routines exist, many of which must be called before the user may begin any calculations. These include:

- → Routines to begin and end the client–server session.
- → Declaration routines to inform CPULib of the pre–defined sets over which distributed data is to be stored. These include elements, faces, edges and nodes, as well as boundary nodes and faces.
- → Routines that inform CPULib which connectivity pointers are needed by the user's application. By default, a file containing the element to node connectivity is read (where the elements may be triangles, quadrilaterals, tetrahedra, prisms, pyramids or hexahedra), as well as boundary information. Since CPULib is able to read in multiblock structured meshes or meshes that have already been partitioned, a list of interface nodes may also be read in. If other pointers are required (for example, the edge–to–node pointer), these are created and partitioned by the CPULib master. However, only a limited number of such pointers are available in the present implementation.
- → A subroutine that tells the CPULib master to partition the pointers declared by the user. At present, Recursive Inertial and Graph Bisection algorithms are available. Overlapping sets are created if required, and an optional Reverse Cuthill McKee algorithm may also be used to reduce the connectivity bandwidth within partitions.
- → A subroutine that aborts the parallel application.

4.2. Inter–Partition Communication

Communication between processors is controlled by a few simple routines:

- → Routines for calculating sums, norms, vector products and global reduction operations — these make use of CPULib's distinction of *owned* and *unowned* subsets to avoid redundant calculations.
- → Routines for initialising data exchanges: each array that is to be updated via message passing at interfaces is first declared by a call to an initialisation routine. Information provided by the user indicates the set over which the data is distributed, as well as the type of exchange appropriate to the array in question. Messages between a given pair of processors may be concatenated by calling this subroutine for each array in question *before* proceeding with either of the procedures given below.

→ Routines for updating interfaces: after initialising the message–passing phase, data from distributed arrays may be exchanged using a single CPULib command. Overlapping of communication and computation is possible by replacing this single call by references to subroutines that perform 'send' and 'receive'–like operations. However, the user is responsible for ensuring that any calculations that make take place between these calls does not depend on the result of the exchange.

4.3. Iterative Techniques

CPULib offers an obvious framework in which to implement well–known iterative methods based on distributed data. The user provides residual evaluations when required and the solution is updated accordingly. Some simple techniques are presently being introduced in collaboration with other European partners. These include:

→ Runge–Kutta methods for integrating discrete PDE's.

→ Krylov methods such as Conjugate Gradient, GMRES and BiCGStab. These may be combined with Newton techniques for non–linear problems (see [4]).

4.4. I/O Routines

Several I/O routines resembling standard FORTRAN commands are provided:

→ Formatted or unformatted files that are to be accessed in parallel may be opened and closed by the user's application. CPULib controls and queues requests to open different files that are assigned identical channels by distinct slaves, although it provides free channels by default.

→ Arrays that are to be distributed may be read as single or ready–partitioned data–sets; these are partitioned if necessary and the relevant parts are automatically sent to the appropriate processor. Such data may also be written into a single file in corresponding forms. Data not associated with any defined set may be broadcast to all partitions; on writing, such data corresponds to that of the root process.

→ Indirection lists that point into a known set may be read and distributed, as well as data defined on the elements of this list.

→ Each slave may also write and read independently – this is useful for debugging purposes, or for advanced users.

4.5. Supporting Routines

Several other supporting routines are included in the **CPU** library. These include:

→ Interfaces with the C functions malloc, ralloc and free, for allocating, reallocating and deallocating blocks of memory. A comprehensive set of memory management tools are also provided. Dynamic array dimensioning may be achieved within Fortran 77 using the POINTER extension. The use of dynamic memory is crucial on parallel machines, since the number of partitions and hence the size of arrays on each processor is not known apriori.

→ Routines for calculating clock times, cpu times and system times

5. Results

In order to demonstrate the effectiveness of CPULib, as well as the use of the different algorithms described in section three, we consider the application of an unstructured explicit Euler code to a three dimensional M6–wing. Since only a relatively small number of processors is used for our tests, a small tetrahedral grid of only 58000 elements is employed; the ratio of partition to interface size is then representative of that which we may expect when solving much larger problems on massively parallel machines. In Table 1, we show the number of cells and nodes that result from partitioning this mesh into N subsets using different overlapping strategies. Although the partitioning algorithm may not be 'optimal', we see that the size of the various sets can increase dramatically. In Table 2 we show the parallel efficiency obtained for the same test cases using different hardware platforms (despite the existence of more efficient message–passing software, we have used PVM version 3.3 in all cases). The speed–ups obtained for the different algorithms imply that the redundant computations on cells are far more important than communication costs, at least for this simple explicit algorithm. The table also demonstrates that overlapping communications and computations can improve the performance of our algorithms on some machines. Space limitations preclude a more detailed examination of the pros and cons of the different approaches. In the present paper, our aim is merely to demonstrate the versatility of the interface treatments within the CPU Library.

As an example of a large–scale CFD application, Figure 3 shows calculated pressure contours and the partitioned surface of an aircraft grid containing 850,000 tetrahedral cells and 156,000 nodes. A Recursive Inertial Bisection algorithm was used for mapping the problem onto eight processors. We note that the automatic partitioning and preprocessing of this mesh takes CPULib less than one minute of real time on the host node of an SP2.

6. Final Remarks

We have demonstrated how it is possible to create a FORTRAN library that abstracts the parallel data–structure from the user's application programme. We believe that such an approach is crucial if parallel computers are to be used effectively in the future. We do, however, consider this library to be a prototype of a much more complete software tool. This would offer a more comprehensive set of partitioning algorithms, indirection pointers, and iterative methods. It would also have the ability to deal with coupled physical problems on independent grids.

REFERENCES

1. P. I. Crumpton and M. B. Giles. Oplus: A parallel framework for unstructured solvers. *Parallel CFD 95*, This procedings.
2. R. Das, J. Saltz, and H. Berryman. *A Manual for PARTI Runtime Primitives, Revision 1.* ICASE, NASA Langley Research Centre, Hampton, USA, May 1993.
3. L. Giraud, P. Noyret, E. Sevault, and V. Van Kemenade. *IPM 2.0 User's Guide and Reference Manual.* CERFACS, Toulouse, France, 1994.
4. F. Guerinino R. Choquet and M. Rudgyard. Towards modular cfd using the cerfacs parallel utilities. *Parallel CFD 95*, This procedings.

N	1	2	4	8	16
RIB	57564/10812	57564/11596	57564/12871	57564/14299	57564/15816
RIB p/o	57564/10812	61420/12245	67813/14648	74240/17296	81144/20170
RIB f/o	57564/10812	65906/13075	79853/16897	95269/21418	113489/26796

Table 1
Total no. of cells/nodes (p/o = partial overlap, f/o = full overlap); M6 wing

N		1	2	4	8	16
SGI Power Challenge	n/o	1.0	1.96	3.22	—	—
	c/o	1.0	1.95	3.88	—	—
	g/o	1.0	1.91	3.36	—	—
	c/o + g/o	1.0	1.93	3.39	—	—
Convex Meta Series	n/o	1.0	1.72	3.21	5.33	—
	c/o	1.0	1.75	3.23	5.77	—
	g/o	1.0	1.69	3.04	5.09	—
	c/o + g/o	1.0	1.70	3.01	5.13	—
IBM SP1	n/o	1.0	2.04	3.96	7.51	12.87
	g/o	1.0	1.97	3.65	5.37	—

Table 2
Speed-up using PVM 3.3 (n/o = no overlap, c/o = communication/computation overlap, g/o = full grid overlap); M6 wing

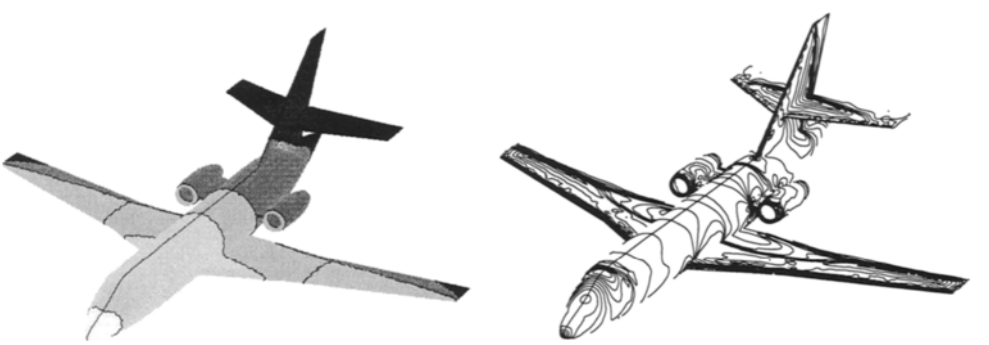


Figure 3. Partitioned grid and pressure contours for the Dassault Falcon

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Host-node client-server software architecture for computational fluid dynamics on MPP computers

Sukumar R. Chakravarthy

Metacomp Technologies, Inc.,
5540 Wembly Avenue, Agoura, California 91301, USA

1. INTRODUCTION

In effectively adapting full-fledged Computational Fluid Dynamics (CFD) codes designed for conventional workstations and supercomputers to Massively Parallel Processor (MPP) computers and in designing new CFD software for them, the host-node and client-server programming paradigms play beneficial roles. They permit the various tasks to be encapsulated in an object-oriented fashion and result in the synergistic use of all available computational resources including high performance MPP computers and workstations with graphics and X-window capabilities, etc. These aspects are discussed below.

1.1. The Typical Hardware Configuration

A typical computing environment today would comprise an MPP computer and workstations linked by a network. They could be in the same building or the MPP computer may reside in a facility removed from the workstations. The MPP computer is assumed to be based on a MIMD (each CPU can execute different instruction streams) distributed memory message-passing architecture. The workstation network would have a common disk farm (accessed via NFS). The MPP computer may or may not have its own massive disk system and its own direct ethernet-type networking capability. At least some of the workstations would have powerful hardware-based graphics rendering and manipulating capability. A "framegrabber" may be attached to the MPP computer directly. The network may also include X-stations as separate from graphics workstations. The graphics workstations would include X-windowing capability. It would be best to develop applications software that can make use of special hardware when they are available but which can effectively cope when not all of the types of computer resources mentioned above are available.

The minimum configuration to be considered for the purposes of this paper comprises the following: one MPP computer, one or more graphics workstation with X-windows, and disks on the the workstation (direct or NFS). We will assume that the amount of main memory available on the MPP computer far exceeds the maximum possible on an individual workstation.

1.2. The Host-Node Approach

Many MPP computers are accessed via a "host" computer (some of them can be accessed via many hosts) which may be a workstation or even an non-MPP supercomputer. Some MPP computers are configured such that one of their "nodes" acts as the host. In the host-node programming model, programs run concurrently on the host as well as on a chosen set of nodes. The programs can communicate with each other by passing messages.

1.3. The Client-Server Perspective

Each type of computer is particularly effective in certain roles, e.g. "number crunching", graphics, disk I/O (input/output), external network communications, etc. In a networked "cooperative" environment, each computer should perform its most effective role as a "service" to the other computers which are not best suited to that role. The resources offering the services are the "server" in a "client-server" perspective. The services are utilized by "clients". For number crunching, the MPP computer is the server in our typical environment. For disk I/O, in the minimum configuration outlined above, the workstation would be the "file server". For an MPP computer without direct ethernet capability, the workstation will also be the network communications server. The workstation will also be the graphics server. When the MPP computer has its own disk farm it can actually be an extremely powerful "NFS server" if configured with appropriate software to perform I/O services. In the client-server perspective applied to the host-node model, the host and the nodes support each other in ways detailed below.

2. THE USER'S VIEWPOINT

To the user, the computational environment that includes an MPP computer should be no different than and no worse than one based only on conventional computers. In fact, it should be better. Resources available should be cost effective. This can best be achieved by avoiding unnecessary duplication of hardware resources. This implies that the memory requirement on the workstation that acts as the host does not have to be same as the memory requirement on the MPP system that is attached to it as a "compute server". Conversely, when large disk farms are available on the host's network, this does not have to be duplicated on the MPP computer. However, this compromises I/O throughput for a job running on the MPP computer because all file I/O will have be channeled through the host (usually sequentially, node by node).

CFD involves problem setup, obtaining the solution and analysing the solution. These three stages can be concurrent and iterative. Problem setup and postprocessing should be performed from an individual user's desk if possible. On-demand visualization, solution analysis, solution tracking and control would also be highly desirable. Algorithm "faults" should be "trappable" so that graceful exit from the computer run is possible. This is particularly important on MPP computers because, otherwise when a floating-point exception occurs on one node (CPU), the entire program may "hang".

2.1. Benefits of host-node model

Software based on the host-node model can distribute the overall work involved with problem setup, solving and anlaysing the results in a transparent way between the host and the

nodes without unduly taxing each type of resource and with minimum effort involved in rewriting existing software. All existing user-interface modules can continue to run on the host. All error-checking on user inputs can be performed by the host. Results can be displayed by the host. The user interface for the analysis software can query the user for his instructions, send messages to the MPP computer, obtain that subset of data that is required for the specific postprocessing need, process this data for visualization or other purpose.

The nodes can communicate information time-step by time-step to the host for display in a window or for being recorded in a disk file. The host program can react to user inputs and interrupts and communicate them to the nodes for proper action. Any single node CPU that encounters a signal as a result of a trapped error can signal the host which, in turn, can send messages/interrupt to all nodes to terminate the entire job.

3. THE APPLICATION DOMAIN

The application domain can range from CFD as as a single discipline through multidisciplinary simulations (e.g. aerodynamics and rigid body dynamics (store separation), aerodynamics and structural dynamics (aeroelasticity), aerodynamics and electromagnetics (design optimization)).

The computational method may use structured grids (single or multiple blocks), unstructured grids (single or multiple blocks), or a hybrid. For multiple blocks, there may exist relative motion between blocks.

The physics being modeled in a CFD computation may describe inviscid flow (Euler equations), viscous flow (Navier-Stokes equations) that may be laminar, turbulent or transitional. The aerothermochemistry may include choices including perfect and real gases, chemical and thermal equilibrium and nonequilibrium. A given simulation will employ some combination of disciplines, gridding approach and physics.

3.1. Benefits of host-node model

The problem definition step involves setting up the geometry, "painting" the boundary conditions and specifying the initial conditions. These can be performed with full user interactivity by programs running on the host. The mesh generation can either be performed on the host or on the nodes depending on the chosen algorithm. Grid generation methods based on solving partial differential equations can easily be mapped to MPP computers. Interzonal connectivities that are not automatically identified by the grid generation process can be computed by preprocessor software on the host. The connectivity and boundary condition information can be communicated to the large scale simulation running on the nodes. Thus, preprocessing and postprocessing software need not be rewritten for the nodes.

When some blocks move with respect to other blocks, the movement may be governed by rigid body dynamics applied to one or more objects in these blocks. The six-degree-of-freedom computations can be performed on the host or on dedicated nodes or on the nodes that are engaged with the numerical flow simulation. The grid redefinitions can be performed on the host along with recomputation of the zonal interconnectivities, etc. and this information can be transferred back to the appropriate nodes.

If the host program is responsible for first reading in user inputs, it can exploit this knowledge to optimize the availability of nodal resources. For example, if the host program

determines that the particular run corresponds to inviscid flow, it only needs to load the nodes with the Euler solver part of a more general software. While this could be done manually by a user, the intelligence can be automatically built into the host software to make it easier for users.

4. GEOMETRIC DOMAIN DECOMPOSITION AND SYNTHESIS

Geometric domain decomposition involves breaking up the computational domain of interest into subdomains, each one of which is assigned to an individual CPU node of the MPP computer. The solution is updated for each subdomain in its assigned CPU and edge information is transferred between subdomains (therefore nodes) via messages. Synthesis involves putting the domains back together.

4.1. Benefits of host-node model

Host-resident software can be easily written to act as on-demand file I/O servers for the nodes. The host-resident files can contain grid and solution restart information that can be transferred to the nodes at the beginning of a computationally intensive run and synthesized back by the host program at the end of the run or whenever a "checkpointing" is desired. Conversely, if the MPP computer is more richly endowed with disk space, the I/O service can be provided by the nodes for pre and postprocessing software on the host.

An intelligent host-resident software module can analyze the problem sizes by scanning the header portions of disk files and choose the appropriate number of MPP nodes. This is achieved in a fashion that achieves the desired turnaround time while minimizing internodal communications. Large problem sizes in each mode reduce communication overheads but increase turnaround time for a given size of the total job. Such host software can also be part of a smart job-queueing system that balances the needs of different users and jobs.

5. FUNCTIONAL DOMAIN DECOMPOSITION AND SYNTHESIS

While geometry-based domain decomposition is the most commonly employed approach, one can exploit other decomposition methods to split the overall problems into pieces, each of which fits a CPU node of an MPP computer. Functional domain decomposition maps the problem space to the domain of functional modules and then maps different modules to corresponding sets of CPU nodes. As an example, for turbulent flow CFD applications, turbulent eddy diffusivities can be computed by nodes assigned for that purpose concurrent with the evaluation of inviscid fluxes by other nodes. As another example nodes could be dedicated for on-demand graphics. Functional domain decomposition, in turn, can use geometric domain decomposition. Synthesis involves putting together the functionally decomposed information content.

5.1. Benefits of host-node model

The host can serve as the functional decomposition controller and based on the user's inputs choose the appropriate division of labor (e.g., physics or turbulence modeling options, etc.). More intelligent software can be built in this way. By allocating the appropriate MPP nodal

resources for the various functional parts of the entire job, faster turnaround can be achieved while maintaining a high level of parallel processing efficiency.

6. APPLICATION DOMAIN DECOMPOSITION AND SYNTHESIS

For multidisciplinary applications, the various application domains can be provided dedicated sets of CPU nodes. This application domain decomposition can exploit functional domain and geometric domain decompositions. Synthesis involves coupling appropriate pieces of information from each domain.

6.1 Benefits of host-node model

For multidisciplinary problems, often the coupling between the disciplines involves far less information transfer than the information transfer between node sets that are computing solutions in the individual disciplines. In this case, the coupling of the disciplines can be performed by the host very effectively without creating load imbalances on the nodes.

7. OTHER BENEFITS OF HOST-NODE MODEL

The host-node model permits users to share a set of nodes between different processes. Therefore space sharing uses of an MPP computer and time sharing are both enabled.

This Page Intentionally Left Blank

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Software Tools for Parallel CFD on Composite Grids *

Peter Olsson, Jarmo Rantakokko, and Michael Thuné [a]

[a]Department of Scientific Computing
P.O. Box 120
S-751 04 Uppsala, Sweden

A library of software tools has been developed for parallel PDE solvers on composite structured grids. The tools have an object-oriented design. As a test case, the tools have been used for the implementation of a realistic problem, giving code that is portable on a large set of serial and parallel computers. This solver has been compared to a code written in plain Fortran 77. The results show that the tools are comparable with plain Fortran code and that they scale very well on a parallel computer with distributed memory.

1. INTRODUCTION

We present COGITO, a set of software tools for composite-grid methods. Code containing calls to the Cogito routines will be portable over a wide range of computers, serial and parallel. Platforms supported are at the moment the software package PVM (cluster of workstations), Intel's NX message passing library (iPSC/2, iPSC/860, and Paragon), and IBM's MPL message passing library (IBM SP1 and SP2). The current implementation of Cogito is in Fortran 77. New parts of Cogito are being coded in Fortran 90.

The tools raise the level of abstraction in the code considerably. Moreover, they make it fairly straightforward to write a composite-grid code which can be executed for example on a multicomputer. This means that the user of Cogito can focus on the numerical method and the mathematical problem instead of on low-level details. Production code is written in a serial fashion, all parallelization aspects are hidden inside the tools. We can develop and debug the code on a single workstation and then move it unmodified to a multicomputer.

Two of the goals for the Cogito project are: (1) to abstract away from computer dependencies, for portability, and (2) to abstract away from low-level representations of the data structures. There are two motivations for (2). One is that advanced data structures, such as composite grids, are not supported in existing programming languages. The other motivation is that we should not need to rewrite the PDE solver if the low-level representation of data is changed.

As a typical application, we consider a 2D compressible Navier-Stokes solver. We compute airflow in an expanding and contracting tube that could be interpreted as a muffler on an exhaustion pipe of a car. The equations are written in general curvilinear coordinates

*The research presented in this paper was supported by the Swedish National Board for Industrial and Technical Development

and the computational domain is covered by a composite grid consisting of five structured rectangular grids. The equations are solved with a Runge-Kutta method and explicit finite differences. We compare code written in plain Fortran 77 with a rewritten version with our tools of the solver and give execution results from different multicomputers. The conclusions are that the code based on our tools is comparable in performance with hard-coded solvers, and scales well on parallel computers of MIMD type with distributed memory.

Cogito has an object-oriented design. This has several advantages. In the present context, information-hiding – leading to portable and easily maintainable code – should be mentioned. Moreover, this kind of design tends to yield a good modularization, and fairly small software parts (operations on classes) that can be combined in a flexible way. Finally, object-oriented software focuses on concepts in the application domain. This increases the readability of the code.

In comparison to recent related work, [5] and [12], we aim at a higher level of abstraction. The work in [3] focus on another class of problems, but their approach is similar to ours. Finally, a research group at LANL is currently implementing classes in C++ for composite-grid methods. Their work is related to ours, and cooperation is being discussed.

2. A LIBRARY OF SOFTWARE TOOLS

The tasks that are handled by Cogito are of two types. One set of tools are on a programmer's level, another set on the numerical analysis level. The tools on the programmer's level concern message passing, management of communication topologies, dynamic arrays, and automatic partitioning and distribution of arrays. The tools on the numerical analyst's level concern grids and grid functions.

Cogito is structured into two layers: one subsystem for the computer related classes (Cogito/Parallel) and one for the grid and grid function classes (Cogito/Grid). Roughly speaking, Cogito/Parallel is intended for the programmer and Cogito/Grid is for the numerical analyst. The classes in the layer Cogito/Grid are based on Cogito/Parallel.

2.1. Parallel tools

Cogito/Parallel handles message passing, data distribution, and management of trees of arrays in distributed memory.

To handle message passing, we have introduced classes *Message* and *Communication Topology*. The class *Message* was developed several years ago, but has not been previously published. The ideas are similar to those of the Message Passing Interface (MPI). A message is created as an object. There are operations for packing data into the message, sending data in various modes, receiving and unpacking data. The class *Communication Topology* [6] handles different communication topologies which are embedded in the physical communication topology of the multicomputer. If it is possible, the neighbors in the logical communication topology are also neighbors in the physical communication topology. At present we have ring, grid, and tree topologies. There are operations for translating between node indices in the embedded topology and physical node identities. There are operations for finding the neighbors and the number of neighbors of a node. A node can also find out if it is in an extreme position of the communication network. For example, the node can be first or last in a ring topology, at the west, east, north, or south

end of a 2D grid, and the root or leaf in a tree topology.

To handle data distribution we have the class *Distribution* which describes how data are to be distributed over the nodes [4]. At present we support the partition shapes slices, rectangles, and boxes which partition across one, two, and three dimensions, respectively. The class Distribution has constructor operations for automatic partitioning for each of these shapes. Moreover, there is an option for automatic choice of the best shape. To be able to make this choice, there are additional classes *Work Map*, *Computer Model*, and *Communication Pattern*. A communication topology is also made automatically and is associated to the distribution. If, for example, a slice partitioning has been made the associated communication topology will be a ring, and if the partitioning is made with rectangles the associated topology will be a 2D grid etc.

The partitioning is based on a strategy, which is analyzed in [9], [10]. It is shown to be adequate in many situations. An improved composite-grid partitioning algorithm is proposed in [11], but has not yet been incorporated into the class Distribution.

To handle the management of trees of arrays in distributed memory we have classes *Array*, *Distributed Array*, and *Directory Tree*. An array is created as an object and there are operations for retrieving the number of dimensions and the size in each dimension. A distributed array is associated to a distribution object. The distributed array object is stored in one of the directories of a directory tree. Each node stores the complete directory structure, but only its own partitions of each array. The class *Distributed Array* inherits from the the class *Array*.

In summary, the classes in Cogito/Parallel give tools for writing low-level multicomputer code. Portability is mainly achieved by providing different implementations of the class Message. Currently, we support the platforms: IBM SP1 and SP2 with MPL, Intel parallel computers with NX, and networks of workstations with PVM.

2.2. Grid tools

Cogito/Grid deals with the management of composite structured grids and grid functions. The task in focus is how to use an existing grid. A grid is a discretization of the computational domain. A composite structured grid is a union of logically rectangular grids patching or overlapping each other. Grid functions are defined on the grids and are used, e.g., to represent the solution of the PDE on the discrete domain.

The classes in this layer are implemented on top of the Cogito/Parallel tools, and the code will thus be portable over the same platforms. Cogito/Grid uses an SPMD programming style, hiding all message passing in the defined operations. The same program can then, without modification, be executed both on a serial computer and a parallel computer of MIMD type with distributed memory.

In the following short code sequences are included to indicate the style of a Fortran program using Cogito. The operations on the classes are implemented as subroutines, having a reference to the object as an argument. The name of the subroutine is a concatenation of the operator and an abbreviation of the class name.

The code sequence below creates a composite-grid object by reading its structure from file, then decides how to partition the grid, and finally reads the grid data distributing it to the owner processors.

```
c     Create and distribute the grid
      call Create_CG(cg,'myFile')
      call Distribute_CG(cg,'normal')
      call Read_CG(cg,'myFile')
```

Grid functions are defined on the grids and thus inherit the same structure and partitioning. To define a grid function u, with four components per grid point, on the entire composite grid created above, the following is sufficient:

```
c     Create grid functions
      call Create_GF(u,'solution',4,cg)
```

The user who wants to write a PDE solver on the Cogito/Grid level of abstraction will mainly operate on grid functions. Currently, only explicit difference methods are supported. A set of arithmetic operations are defined for the class. In particular, there are methods for applying difference operators to the grid functions.

```
c     Differentiate u
      call DiffOp_GF(u,du,il,ir,w)
```

The parameters *il* and *ir* define the operator widths and w the difference weights. If the grid is partitioned the operation will be done in parallel. First, partition boundary data are sent to neighbor nodes. While waiting for the data to arrive, inner points are differentiated in order to overlap computations and communication. Finally, the partition boundaries are differentiated.

Grid functions defined on a composite grid must be interpolated between the component grids. Interpolation is performed through the operation *Interpolate*. Assume that the grid function u is to be interpolated from grid $G1$ to grid $G2$. The interpolation consists of two phases. In the first phase, the processors owning relevant partitions of $G1$ compute interpolation values. These values are gathered using a table (defined in *Create_CG* and initiated in *Distribute_CG*) and sent to the processors who have the corresponding partitions of $G2$. The second phase consists in the latter processors receiving the interpolation values, and updating u on $G2$. The operation *Interpolate* has a *mode* parameter, which makes it possible to execute each phase in a separate call. With *mode* = *pre* the first phase is executed, with *mode* = *post* the second one. This allows for overlap of communication and computation:

```
c     Interpolate u
      call Interpolate_GF(u,pre)
      ...computations not needing interpolation data...
      call Interpolate_GF(u,post)
```

Also boundary conditions are implemented by using the arithmetic operators in the class *Grid Function*. A subdomain of the grid can be defined and an identifier of this subdomain is passed as an argument to the arithmetic operators. The operators are performed only on the specified subdomain.

In [7] the tools on the Cogito/Grid level are described more in detail.

3. EVALUATION OF THE TOOLS

Three issues are of interest for the evaluation of the tools:

- Portability
- Single node performance
- Parallel performance

To show portability we have used three different computers, SUN Sparc 10, IBM SP2, and Intel Paragon XP. The SUN Sparc 10 is a serial workstation, the IBM SP2 is a cluster of workstations connected with a cross-bar switch, and the Intel Paragon XP is a dedicated parallel computer with the nodes in a 2D mesh.

For the second issue, serial execution speed, we have compared serial code written in plain Fortran 77 with code using the grid tools. We have implemented a two-dimensional compressible Navier–Stokes solver on a multi-block grid with five blocks. The solver uses centered, second order accurate differences in space and Runge-Kutta time-marching. This program was originally written for a one-processor machine [2]. We have rewritten the code making use of the Cogito/Grid tools. In our experiments, the solver was executed on a single node on the three different platforms, see Figures 1 to 3. All grids are quadratic and of the same size. Neither of the codes was tuned for optimal performance on any specific computer and only standard compiler optimizations were used.

The results in Figures 1 to 3 show no clear tendency in difference between the Cogito and the plain Fortran 77 version. Other experiments with a similar solver [7] indicate the same results. However, the results for the three cases show that improvements are possible. Memory management and processor utilization are two important factors affecting the performance. Information about processor type, cache memory, etc could be included in the class *Computer Model*, which is part of the Cogito/Parallel subsystem. The Cogito/Grid tools could then be modified so that they adapt to these details (by loop-unrolling, vector-length adjustment, loop-ordering, addressing modes, etc). A source of differences in performance between the two codes is the programming style. The object-oriented approach tends to yield many calls to relatively small operations in the classes. By adding compound operations to the class *Grid Function* the performance of the Cogito based code could be improved.

For an evaluation of the parallel performance of the tools, issue three, the performance of the Cogito based Navier–Stokes solver was studied. (The corresponding hard-coded version was not parallelized.) As a measure of parallel performance, we used sizeup [8]. This means increasing the problem size, to maintain fixed execution time when the number of processors grows. Sizeup is defined as parallel work divided by serial work. As the problem size per processor is approximatively kept constant, sizeup is less dependent

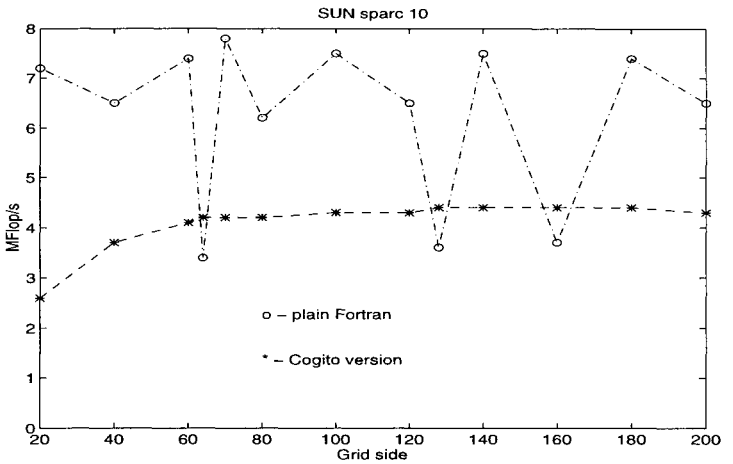


Figure 1. Performance, SUN sparc 10

of the single node performance and hence of the node architecture and of different compiler optimizations. In our experiments, the one-processor problem size is $50*50$ points in each block. The results, see figure 4, show that the Cogito-based code scales very well.

4. SUMMARY AND CONCLUSIONS

We have presented a software package for construction of portable, parallel PDE solvers. The package decouples different program components and gives support for complex data structures such as composite grids. Moreover, we have described an implementation showing that the tools can be used for realistic problems and that the object-oriented design raise the level of abstraction which increases the readability of the code. Execution timings show that the tools are comparable in performance with hard-coded solvers and that they scale very well on parallel computers of MIMD type with distributed memory.

Cogito is still under development. The single node performance of the routines needs to be improved. Moreover, classes for I/O management and out-of-core data storage need to be added. Finally, a third level of abstraction, Cogito/Solver, is under development. These tools will separate the numerical method from the boundary conditions and the mathematical problem.

REFERENCES

1. G. Chesshire, W. D. Henshaw, *Composite meshes for the solution of partial differential equations*, J. Comp. Phys., 90 (1990), pp. 1-64.
2. R. Enander, *Grid patching and residual smoothing for computations of steady state solutions of first order hyperbolic systems*, Thesis, Dept. of Scientific Computing, Uppsala University, Uppsala, 1991.

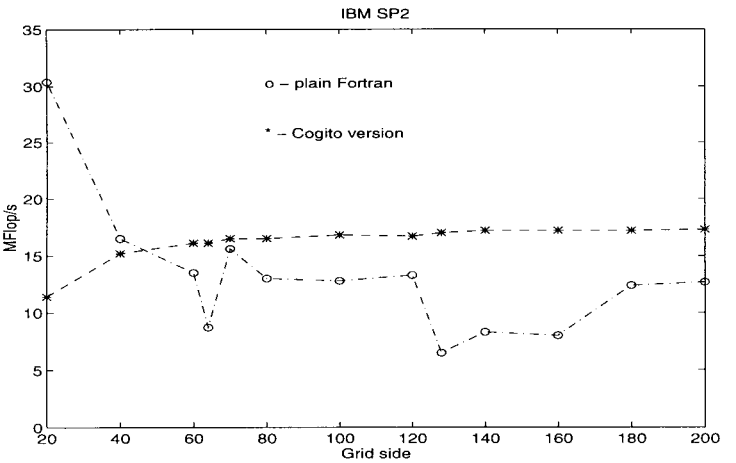


Figure 2. Single node performance, IBM SP2

3. W. Gropp, B. Smith, *Scalable, extensible, and portable numerical libraries*, in Proceedings of the Scalable Parallel Libraries Conference, pp. 87–93, IEEE, 1994.
4. K. Högström, M. Thuné, *Portability and data structures in scientific computing — object-oriented design of utility routines in Fortran*, in Parallel Computing: From Theory to Sound Practice (W. Joosen and E. Milgrom, Eds.), pp. 585-588, IOS Press, Amsterdam, 1992.
5. M. Lemke, D. Quinlan, *P++, a parallel C++ array class library for archi-tecture-independent development of structured grid applications*, ACM SIGPLAN Notes, 28 (1993), pp. 21-23.
6. P. Olsson, *Object-oriented design of Fortran routines for the management of communication topologies on a hypercube*, Institute of Technology, Report UPTEC 91 130E, Uppsala University, 1991.
7. J. Rantakokko, *Object-oriented software tools for composite-grid methods on parallel computers*, Report 165, Dept. of Scientific Computing, Uppsala University, Uppsala, 1995.
8. X. Sun, J.L. Gustafson, *Towards a better parallel performance metric*, Parallel Computing, 17 (1991), pp. 1093-1109.
9. M. Thuné, *A partitioning strategy for explicit difference methods*, Parallel Computing, 15 (1990), pp. 147-154.
10. M. Thuné, *Straightforward partitioning of composite grids for explicit difference methods*, Parallel Computing, 17 (1991), pp. 665-672.
11. M. Thuné, *A partitioning algorithm for composite grids*, Parallel Algorithms and Applications, 1 (1993), pp. 69-81.
12. R. D. Williams, *DIME++: A language for parallel PDE solvers*, CCSF, Report CCSF-29-92, Caltech, Pasadena, 1993.

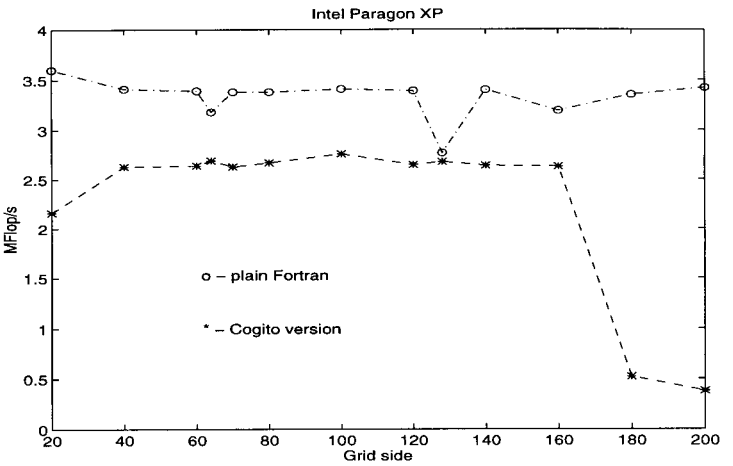


Figure 3. Single node performance, Intel Paragon XP

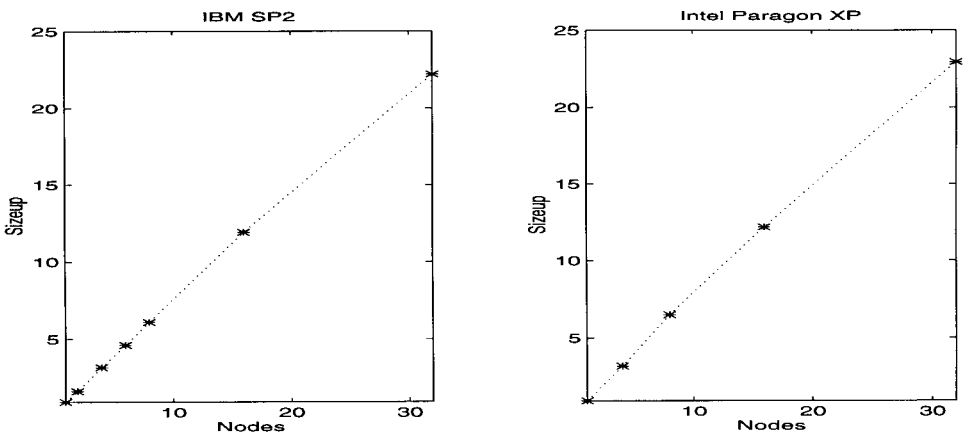


Figure 4. Parallel Performance, Sizeup