Luciano TARRICONE
Alessandra ESPOSITO

grid computing

FOR ELECTROMAGNETICS

CD-ROM INCLUDED

Grid Computing for Electromagnetics

For a listing of recent related titles, turn to the back of this book.

Grid Computing for Electromagnetics

Luciano Tarricone
Alessandra Esposito

Artech House, Inc.
Boston • London
www.artechhouse.com

Library of Congress Cataloguing-in-Publication Data
Tarricone, Luciano.
Grid computing for electromagnetics/Luciano Tarricone, Alessandra Esposito.
p. cm.
Includes bibliographical references and index.
ISBN 1-58053-777-4 (alk. paper)
1. Computational grids (Computer systems) 2. Electromagnetism–Data processing.
I. Esposito, Alessandra. II. Title.

QA76.9.C58T37 2004
004'.36—dc22

2004053827

British Library Cataloguing in Publication Data
Tarricone, Luciano
Grid computing for electromagnetics. –(Artech House electromagnetics library)
1. Electromagnetism 2. Computational grids (Computer systems) I. Title II. Esposito, Alessandra

621.3'0285436

ISBN 1-58053-777-4

Cover design by Igor Valdman

© 2004 ARTECH HOUSE, INC.
685 Canton Street
Norwood, MA 02062

All rights reserved. Printed and bound in the United States of America. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing from the publisher.

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Artech House cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

International Standard Book Number: 1-58053-777-4

10 9 8 7 6 5 4 3 2 1

Contents

Acknowledgments *xi*

Introduction *xiii*

Grid Computing: What Is It? *xiii*
Grid Computing: Who Is Who? *xv*
Grid Computing: An Opportunity for Electromagnetics Research *xv*
How to Read This Book *xvii*
A Final Note *xviii*
References *xviii*

CHAPTER 1
General Concepts on Grids 1

1.1 Introduction 1
1.2 Parallel and Distributed Architectures 2
1.3 Parallel and Distributed Topologies 5
1.4 Parallel and Distributed Programming 7
1.4.1 Message Passing 8
1.4.2 Shared-Memory Programming 9
1.4.3 Concluding Remarks: Programming Paradigms and Parallel Architectures 10
1.5 Performance Assessment 10
1.6 Web Computing 11
1.7 Computational Grids 14
1.7.1 Introduction 14
1.7.2 What Is a Grid? 15
1.7.3 Grid Architecture 17
1.7.4 Grid Middleware 19
1.7.5 Applications 20
1.8 Conclusions 21
References 21

CHAPTER 2
Enabling Technologies and Dedicated Tools 23

2.1 Introduction 23
2.2 Enabling Technologies: Object Orientation 24
2.2.1 Object Orientation for Software Engineering 24
2.2.2 Object Orientation for Enabling Technologies 25
2.2.3 CORBA 26

2.2.4 Java 27
2.2.5 Object Orientation and Electromagnetic Simulators 28
2.2.6 Conclusions 29
2.3 Dedicated Tools: Grid Middleware 30
2.4 The Globus Toolkit: An Overview 30
2.5 The Globus Toolkit: The Globus Security Infrastructure 31
2.5.1 Authorization 32
2.5.2 Mutual Authentication 33
2.5.3 Single Sign On and Delegation 35
2.5.4 Other Services 37
2.6 The Globus Toolkit: The Resource Management Pillar 38
2.7 The Globus Toolkit: The Information Services Pillar 42
2.7.1 MDS Directory Service: Lightweight Directory Access Protocol 43
2.7.2 MDS Information Model 43
2.8 The Globus Toolkit: The Data Management Pillar 46
2.8.1 Distributed Data Access and Management 46
2.8.2 Dataset Replicas Services 47
2.8.3 Conclusions 48
2.9 The Globus Tools API 48
2.10 The MPI with Globus 49
2.11 Dedicated Tools: Economy-Driven RM in Grids 51
2.12 Web-Based Technologies and Projects 51
2.13 Grid-Enabled HTC: Condor-G 53
References 53

CHAPTER 3
Building Up a Grid 57

3.1 Introduction 57
3.2 Recalling Globus Basic Concepts 58
3.3 Setting Up the Environment 60
3.3.1 Hardware Requirements 60
3.3.2 Software Requirements 60
3.3.3 Setting Up the Network 60
3.3.4 Before Installing Globus 61
3.4 Globus Installation 62
3.4.1 Downloading the Package 62
3.4.2 Installing the Toolkit 63
3.5 Globus Configuration 64
3.5.1 Authorization 65
3.5.2 Authentication 66
3.5.3 Using the Globus CA 66
3.5.4 Using a Local CA 68
3.6 Services Start Up 72
3.6.1 Resource Management 72
3.6.2 Information Services 72
3.6.3 Data Management 73
3.7 Introducing a New User to the Grid 74

3.7.1 Client Side 74
3.7.2 Server Side 74
3.8 Globus-Relevant Commands to Use the Grid 74
3.8.1 Authentication 75
3.8.2 Resource Management 75
3.8.3 Information Services 78
3.8.4 Data Management 80
3.9 Developing Grid-Enabled Applications 82
3.9.1 An Example with Globus API 83
3.10 Message Passing in a Grid Framework 85
3.11 Summary and Conclusions 87
References 87

CHAPTER 4
Applications: FDTD with MPI in Grid Environments 89

4.1 Introduction 89
4.2 The FDTD Approach: Theoretical Background 89
4.2.1 Yee's Algorithm 89
4.2.2 Stability of the Algorithm 92
4.2.3 Numerical Dispersion 92
4.2.4 Excitation and Absorbing Boundary Conditions 93
4.2.5 CPU Time and Memory Requirements 95
4.3 Parallel FDTD 96
4.3.1 A Simple and Portable Parallel Algorithm 96
4.4 Migration Toward Computational Grids 108
4.4.1 Introduction 108
4.4.2 Practical Guidelines 109
4.4.3 Pthread Libraries and MPICH-G2 110
4.5 Numerical Performance 111
4.5.1 Performance Evaluation of Parallel Distributed FDTD 111
4.5.2 MPICH-G2 Performance Evaluation 112
4.5.3 Benchmarking Parallel FDTD on a Grid 115
4.6 Remarkable Achievements 116
4.7 Conclusions 117
Acknowledgments 117
References 117

CHAPTER 5
CAE of Aperture-Antenna Arrays 121

5.1 Introduction 121
5.2 Numerical Techniques for the Analysis of Flange-Mounted Rectangular Apertures 123
5.2.1 Theoretical Background 123
5.2.2 Approaches Based on Waveguide Modes 125
5.2.3 Approaches Based on Gegenbauer's Polynomials 127
5.3 A Tool for the CAE of Rectangular Aperture Antenna Arrays 128
5.3.1 Evaluation of the Horns' Scattering Matrix 129

5.3.2 Evaluation of the Aperture Array's Scattering Matrix 130
5.3.3 Evaluation of the Scattering Matrix at External Ports 132
5.3.4 Evaluation of the Radiation Pattern 134
5.4 Parallel CAE of Aperture Arrays 135
5.4.1 Preliminary Analysis 136
5.4.2 Parallelization 139
5.4.3 Results on MIMD Supercomputing Platforms 142
5.5 Migration Toward Grid Environments 144
5.5.1 Supporting Cooperative Engineering with GC 145
5.6 Conclusions 150
Acknowledgments 151
References 151

CHAPTER 6
Wireless Radio Base Station Networks 153

6.1 Introduction 153
6.2 Foundations of Cellular Systems 154
6.2.1 General Considerations 154
6.2.2 Frequency Reuse 155
6.2.3 Capacity and Traffic 157
6.2.4 How a Cellular System Connects Users 158
6.2.5 BS Antennas 158
6.3 Key Factors for Current and Future Wireless Communications 160
6.3.1 Power Control 160
6.3.2 Managing with More and More Users 161
6.3.3 System Standardization and Interoperability 161
6.3.4 Concerns in the Public Opinion 162
6.4 Planning Wireless Networks 162
6.5 An Integrated System for Optimum Wireless Network Planning 163
6.5.1 Overview of the System 164
6.6 A Candidate Architecture for an Effective ISNOP 169
6.7 GC and Its Role in the ISNOP 170
6.8 Wireless Network Planning with GC 170
6.8.1 Data Communication with GC in a Simplified ISNOP 173
6.8.2 ENC Module Simulation 178
6.9 Conclusions 180
Acknowledgments 181
References 181

CHAPTER 7
Conclusions and Future Trends 183

7.1 GC: Benefits and Limitations 183
7.2 GC Trends 184
References 185

APPENDIX A
Useful UNIX/Linux Hints 187

A.1 UNIX/Linux Operating System: An Overview 187
A.2 UNIX/Linux: The Architecture 188
A.3 The File System 188
A.3.1 Introduction 188
A.3.2 File System Relevant Commands 189
A.3.3 Pathnames 191
A.3.4 System Calls for File Management 192
A.3.5 Permissions 192
A.4 Processes 193
A.5 Administration 194
A.6 The Shell 194
A.6.1 Introduction 194
A.6.2 Background Command Execution 196
A.6.3 Redirection 196
A.6.4 Pipes 197
A.6.5 Environment Variables 197
References 198

APPENDIX B
Foundations of Cryptography and Security 199

B.1 Introduction 199
B.2 Confidentiality and Cryptography 200
B.3 Digital Signature 202
B.4 Certificates and Certification Authorities 203
References 205

APPENDIX C
Foundations for Electromagnetic Theory 207

C.1 Maxwell's Equations in the Time Domain 207
C.2 Helmholtz and Dispersion Equations 208
C.3 TE and TM Modes 209
C.4 Fourier Representation of Green's Functions 210
C.5 The Far-Field Approximation 212
Reference 213

APPENDIX D
List of Useful Web Sites 215

Glossary 217

List of Acronyms 227

Selected Bibliography 233

About the Authors 239

Index 241

Acknowledgments

This book is intended as a guide to the use of grid computing, an emerging branch of information technology for researchers involved in electromagnetics. It has a practical orientation and aims at allowing researchers to learn how to set up a computational grid, how to run electromagnetic applications on it, and how to use grid computing to identify new and promising perspectives for their research. The book could also be adopted as a text book in advanced courses of applied electromagnetics.

Usually, a book is the result of several years of studies, teaching, research, investigations, and discussions, and this book is no different. In the most exciting cases, it is also the starting point of new efforts, and we hope this is the case! Therefore, it is quite often an achievement the authors share with the colleagues, students, and friends who stimulated and encouraged their thoughts. Among them, Luciano wants to acknowledge the colleagues Mauro Mongiardo and Roberto Sorrentino at the University of Perugia, Italy, and Guglielmo d'Inzeo at the University of Rome, La Sapienza, Italy. Alessandra is grateful to Giuseppe Vitillaro for his enthusiastic discussions on current and future scientific computing. Both the authors thank Professor Peter Excell at the University of Bradford, United Kingdom, for his revision of parts of the book.

The authors also want to express their deep gratitude to their parents, who more than anybody else have supported this long pathway, since the very beginning.

The authors want to remember now the memory of Salvatore, who left them just as this book was being concluded.

Finally, the authors dedicate this work to their children, Silvia and Edoardo, whose happy laughing has accompanied this long journey.

Introduction

Grid Computing: What Is It?

The continuous progress in scientific research is itself an explanation of the insatiable demand for computational power. On the other hand, one of the results of scientific progress is the availability of more and more powerful computer platforms. This self-feeding cycle is pushing our search for knowledge towards very challenging investigations, and parallel computing nowadays plays an important role in this scenario. This is especially driven by the present-day enhancement in distributed computing, which has produced a substantial reduction in the costs of effective supercomputing facilities.

Another emerging trend, due to the improvement of distributed information technologies (IT), is the acceleration of research and development processes towards concurrent and cooperative engineering. Daily workflows in academic and industrial activities are more and more based on interaction among remote entities, which in some cases are physical people and in others are agents or facilities embedding value-adding procedures. An IT infrastructure is, most of the time, the core of such processes.

In the last decade, these important evolutions have been accompanied by the so-called *Internet revolution* and the boom in Web applications. The extraordinary perspectives opened by the Web have reinforced the momentum towards process integration and cooperative computing. Consequently, joining together supercomputing facilities and the world of Web-based tools seems to be the key feature to opening new perspectives in industrial and scientific computational processes, and an emerging technology is being proposed as the most natural way to pursue such a goal: grid computing (GC).

The technology of GC has led to the possibility of using networks of computers as a single, unified computing tool, clustering or coupling a wide variety of facilities potentially distributed over a wide geographical region, including supercomputers, storage systems, data sources, and special classes of devices, and using them as a single unifying resource (computational grid). The concept started as a project to link supercomputing sites but has now grown far beyond its original intent, opening new scenarios for *collaborative engineering, data exploration, high-throughput computing* (*HTC*), *meta application*, and *high-performance computing* (*HPC*).

Collaborative (or *cooperative*) *engineering* means providing engineers and researchers with tools for cooperating online. These tools allow them to share remote resources, modify them, and design, implement, and launch applications in cooperation. In such a context, a grid can be seen as a global production

environment, where distributed systems can be prototyped and tested. The flexibility of grids makes this system dynamic and configurable. Via the grid, researchers are able to rapidly modify their products in order to adapt them to the changes of underlying environments, infrastructure, and resources.

Data exploration is particularly critical when dealing with huge amounts of data and their access from remote sites is needed. Several research fields, such as climate analysis and biological studies, require the storage and accessing of data up to the terabyte or petabyte range. In these cases, data are distributed on a number of remote sites and then accessed uniformly. Taking this further, redundancy can help in improving access performance and reliability. Redundancy is obtained by creating replicas of data sets, increasing performance by accessing the nearest data set. Services to manage distributed data sets and replicas are central in grid computing.

HTC applications require large amounts of computational power over a long period of time. Examples of HTC applications are large-scale simulations and parametric studies. HTC environments try to optimize the number of jobs they can complete over a long period of time. A grid allows exploitation of the idle central processing unit (CPU) cycles of connected machines and use of them for HTC applications.

*Meta application*s are applications made up of components owned and developed by different organizations and resident on remote nodes. A meta application is usually a multidisciplinary application that combines contributions from differently skilled scientific groups. A meta application is dynamic (i.e., it may require a different resource mix depending on the requirements of the current problem to solve). In addition, research teams are able to preserve their property on their own application components, by granting usage through a recognized brokering entity.

HPC applications are, by definition, those that require a huge amount of power (relative to the norm for contemporary computers), usually over a short period of time. This is the primary field for which grids were conceived, as a direct evolution of parallel and distributed processing concepts. Scientific simulations for weather forecasting or astrophysics research are examples of applications requiring huge computational power. The scope of these simulations is limited by the available computational power, even when using supercomputers. Grids allow scientists belonging to different organizations to join their computational resources and acquire amounts of computational power otherwise unaffordable.

All of these applications are supported by GC in a *secure* framework that is *Web compliant* and open to *heterogeneous* platforms and systems. When we mention *security*, we refer to the capability of guaranteeing that the owner of a resource can, at any moment, establish who can access the resource, when, and for what. *Web compliance* is the ability to develop or use applications that take full advantage of recent technologies supporting multimedia applications over the Web. Finally, when we cite *heterogeneous* environments, we refer to the ability to bypass all of the obstacles represented by the coexistence of several architectures, operating systems, programming languages, networking protocols, software methodologies, and technologies.

All of these applications are exemplified in this book. We suggest reading Chapter 4 for an HPC application, Chapter 5 for an application focused on collaborative engineering and meta applications, and Chapter 6 for an application oriented

to data exploration and HTC. Should you be interested just in one of the mentioned areas, you may want to read further in the Introduction, where we suggest how to read the whole book or parts of it.

Grid Computing: Who Is Who?

GC is mature and is attracting large companies, boards, and research centers. For instance, IBM is building the Distributed Terascale Facility (DTF), with $53 million funding from the National Science Foundation (NSF) [1]. Examples of working grid applications can be found in different scientific disciplines. In the field of distributed supercomputing, one of the primary areas in which GC has sparked interest is in large-scale sequence similarity searches. An individual sequence similarity search requires little computation time, but it is common for researchers to perform such searches for thousands or even tens of thousands of sequences at a time. These searches, each dependent on the others, can be spread across as many hosts as are available. The storage and exploitation of genomes and of the huge amount of data coming from post genomics puts a growing pressure on computing tools—such as databases and code management—for storing data and data mining. Genomic research needs, together with requirements coming from other disciplines, gave place to the DataGrid project [2]. This European initiative joins researchers coming from European Organization for Nuclear Research (CERN), European Space Agency (ESA), and other outstanding European scientific centers and is actively fostered by the European Union. It is focused on building up an international grid to store large volumes of data and to provide a uniform platform for data mining. This helps researchers from biological science, Earth observation and high-energy physics, where large scale, data-intensive computing is essential.

Other interesting grid applications are those oriented towards the promotion of synchronous cooperation between persons. The Access Grid [3] is focused on online collaboration through audio/video conferencing. The Astrophysics Simulation Collaboratory (ASC) portal [4] allows users to form virtual organizations over the Internet. People belonging to a virtual organization access the grid to collaboratively assemble code, start-stop simulations, and update and access a repository of simulation components shared with their remote colleagues. In Chapter 2 of this book and in [5], more thorough lists of projects involving grids are to be found. In this introductory context, it is interesting to recall that the reference point for grid communities is the Global Grid Forum (GGF) [6]. The GGF coordinates a growing number of research groups cooperating to ratify community standards for grid software and services and to develop vendor- and architecture-independent protocols and interfaces.

Grid Computing: An Opportunity for Electromagnetics Research

Though the community of electromagnetics (EM) research has been only peripherally interested in GC until now, several practical EM applications can immediately take advantage of GC.

An excellent example is the use of HPC for CPU-demanding tasks, such as the ones using finite-difference time-domain (FDTD) codes for human-antenna interaction. This is a typical CPU-intensive application, quite amenable to parallel computing. Until now, its solution with parallel computing has required the use of costly parallel platforms to achieve effective performance. GC, however, offers a low-cost supercomputing environment, which can be dynamically arranged in order to fulfill the requirements of a specific problem. This is a relevant point, as one of the severe limitations to the diffusion of parallel computing has been the affordability of platforms to achieve high performance.

Another major example is in the computer-aided engineering (CAE) of microwave (MW) circuits and antennas. In such a case, GC allows the integration of design and analysis capabilities in a secure, Web-enabled, high-performance environment.

Consider, for instance, the development of complex MW circuits or antennas composed of several parts, each requiring specific simulation approaches. Suppose also that several research groups are in charge of developing parts of the whole circuit and that some groups are interested in selling, via the Web, their contribution to the other cooperating groups. In addition to allowing the cooperative development of the project and ensuring a high-performance no-cost environment, GC acts as a *broker*, regulating such interactions, even managing payments (when requested) and commercial transactions. The same GC infrastructure can also support the dominant numerical effort typically required by optimization cycles, which at the moment represent one of the crucial steps for achieving adequate performance and yields. Optimization is especially hard when a satisfactory tradeoff must be sought between circuit performance and manufacturing issues, leading to the integration of EM, thermal, mechanical, and economic specifications. In such a case, which is often the bottleneck of industrial processes, GC can represent an appealing and affordable answer to the need of concentrating cooperative/integrated engineering and supercomputing in a single framework.

Finally, an attractive example is represented by the optimum design and planning of wireless networks, an area experiencing a booming growth. We refer, for instance, to the problem of identifying optimum locations and electrical parameters (e.g., tilting, power levels, and orientations) for radio base station antennas, so that a high quality of service, coverage, and traffic management is guaranteed, along with compliance with safety standards for human exposure to EM fields. In such a case, the traditional expertise of EM researchers, such as radio propagation and antenna skills, must harmonize with telecommunication and optimization requirements, in a single integrated information system that also uses digital cartography and high-performance visualization. This very multidisciplinary and complex challenge involves problems typical of cooperative engineering, meta applications, and supercomputing, as well as severe problems of management of large amounts of data, distributed on a geographical basis and critical from a security point of view. Even in this case, GC is the compact answer to the wide variety of enumerated problems and requirements.

The three examples mentioned (FDTD for human interaction, CAE of circuits/antennas, and design/management of wireless networks) are the application areas we focus on in this book. They represent, in our opinion, only some of the

interesting demonstrations of the exciting perspectives opened by GC for EM research: many other challenging opportunities for EM scientists are just around the corner.

How to Read This Book

Different users will have different needs, and so it is appropriate that we give some guidance on approaches in typical cases. Thus, we now give a very short description of how the book is structured and some suggestions on reading it, permitting the skipping of parts some readers may not be interested in or may want to read in a latter step.

In this book, we introduce the interested reader to the use of GC in computational EM (CEM). The book is oriented towards practical applications and aims at enabling the beginner to build up a grid, install or migrate his or her applications, and run them. Therefore, in Chapter 1, we propose general concepts about grids. In Chapter 2, we give a short overview on Globus, a fundamental tool for grid implementation. In Chapter 3, we summarize the main steps in building up a grid. In the next chapters, we propose three EM applications. Chapter 4 deals with the use of GC for parallel FDTD; Chapter 5 deals with the use of GC for cooperative CAE of rectangular aperture array antennas; and Chapter 6 deals with the use of GC for optimum planning, managing, and monitoring of wireless radio base station networks. Finally, in Chapter 7, we discuss advantages and limitations of GC for CEM, and draw some conclusions. A CD-ROM is enclosed, with some sample code for the several application areas. The CD-ROM also includes all of the software needed to build up a grid, assuming that you have at least some UNIX-enabled PCs connected to the Internet.

The reader interested only in building up a grid can directly access Chapter 3 and could substantially benefit from the support of a system engineer with expertise in UNIX systems and networking. This is even more relevant if either the reader wants to use versions of the suggested software different from the ones enclosed in the CD-ROM or platforms adopt different operating systems from the ones to which we refer.

The reader interested only in one of the three proposed EM applications can, in principle, skip the other applications. For instance, the reader interested in wireless networks (Chapter 6) can skip Chapters 4 and 5 without affecting the readability of the text. Nonetheless, it is worth noting that the order in which applications are discussed follows a rationale. Indeed, the FDTD application reported in Chapter 4 is an example of GC for HPC. The application reported in Chapter 5 (CAE of antennas) is an example of GC for HPC *and* cooperative engineering. The application in Chapter 6 (wireless networks) is an example of GC for HPC *and* cooperative engineering *and* data management. Consequently, Chapter 5 omits details on HPC and refers back to Chapter 4 for this part of the application. Equivalently, Chapter 6 omits details on HPC and cooperative engineering, referring to Chapters 4 and 5, respectively, for these parts.

We have also prepared a glossary, where the majority of terms with a technical and scientific relevance are succinctly explained. We have taken special care over

terms coming from the IT and telecommunication areas, trying to propose simple, yet rigorous, definitions. Finally, you can also find an alphabetical list of the adopted acronyms. We hope this renders reading the book easier.

A Final Note

As reference tool for computational grids, the book refers to Globus Toolkit (GT) 2.2.4, which is also attached in the CD-ROM. New Globus versions are continuously being published. Typically, new versions are *backward compatible*: they usually add new facilities, preserving the existing ones. This seems to be confirmed also with version 3, the forthcoming version, with the exception of a simplification of the installation procedure and heavier software requirements to install GT. In conclusion, the CD-ROM-attached GT 2.2.4 version renders the material nonevanescent, and, in general, the only part strictly tied with that version is the part of Chapter 3 describing its installation.

References

[1] http://www.ibm.com/grid.

[2] DataGrid Project Home Page, http://www.eu-datagrid.org.

[3] Access Grid Project Home Page, http://www.accessgrid.org.

[4] ASC Portal Home Page, http://www.acsportal.org.

[5] Baker, M., R. Buyya, and D. Laforenza, "The Grid: International Efforts in Global Computing," *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, Italy, 2000.

[6] GGF Home Page, http://www.gridforum.org.

CHAPTER 1

General Concepts on Grids

1.1 Introduction

Computational grids find their origins and background in the field of HPC, with the preeminent goal of linking supercomputing sites and optimally exploiting CPU time available through a wide area multidomain networking connection. It is a common situation, indeed, that on a certain node, at a given instant, a strong computational effort must be sustained, while huge CPU-time amounts are left idle on remote sites. The existence of a *pervasive intelligence* could monitor the status of each processor, assigning CPU power where needed in a right-sized fashion, thus reducing as much as possible idle CPU times and allowing controlled access to large-scale CPU facilities. The role played by the evoked pervasive intelligence is starred by what we call now a *computational grid*.

After (or, more realistically, while) achieving the goal of facilitating and improving HPC, grids have naturally evolved: the same idea of controlled and optimized management of available distributed CPU power has been extended to the more general concept of *resource* management. As you may need CPU time at a certain moment, you may also want to access a remote database, store large amounts of data on remote storage systems, and access software services and electronic instruments. Grids are in charge of enabling users to do this.

In such a new and variegated context, which evolves along with the impressive Internet revolution and its Web facilities, grids have grown far beyond their early conception, representing the natural melting pot of distributed systems, networking, security, parallel computing, and Web tools.

Consequently, the ideal profile of a researcher to be involved in the migration of scientific applications towards the multidisciplinary context of grid computing should lie on a basic knowledge of parallel and distributed computing, supported by specific skills covering the cultural areas mentioned earlier.

In this chapter, a synoptic overview is given on parallel and distributed computing, with some historical and perspective discussions on architectures and programming paradigms. The important issues of architecture topologies and performance evaluation are described. The reader is given a very brief summary, aimed more at focusing on the most relevant arguments than on discussing them in an exhaustive fashion. Later on, the same analysis is proposed for the area of Web technologies, introducing several themes that are more thoroughly described in the following chapters. On such bases, an introductory description of grid technology is finally proposed, identifying the main possible applications as well as the services and the candidate architecture to support them.

1.2 Parallel and Distributed Architectures

Traditional computers follow the universal model of von Neumann [1] (see Figure 1.1). This model describes the functioning of every computer machine. A CPU processes sequentially instructions stored in memory. The CPU consists of two main components, the control unit (CU) and the arithmetic logical unit (ALU). The CU is responsible for decoding instructions fetched from memory in a CPU local register, labeled instruction register (IR) in the figure. The ALU is responsible for executing them. Input data are moved from memory to the ALU local register, labeled data register (DR) in the figure. Output data are moved from CPU to memory. Input/output (I/O) devices are controlled by the CPU and provide interaction with end users.

The model describes the computer as being made up of three components:

- Memory;
- CPU;
- I/O devices.

Memory provides storage space to contain instructions and data. The CPU processes program instructions by using two units, the CU and the ALU. I/O devices (e.g., video or printer) provide the interface with end users. Computer components interact with one another by exchanging data and signals through a fast communication line (called a *bus*).

The computer executes instructions coded in a specific language and listed in a program that is stored in memory. Processing is carried out serially by performing the following actions:

1. Loading the next program instruction from memory to a special register local to the CPU;
2. Decoding the loaded instruction;
3. Loading data and executing.

Step two is performed by the CU, which interprets the current instruction and sends control signals to the ALU, so that it performs the operations as requested by the loaded instruction.

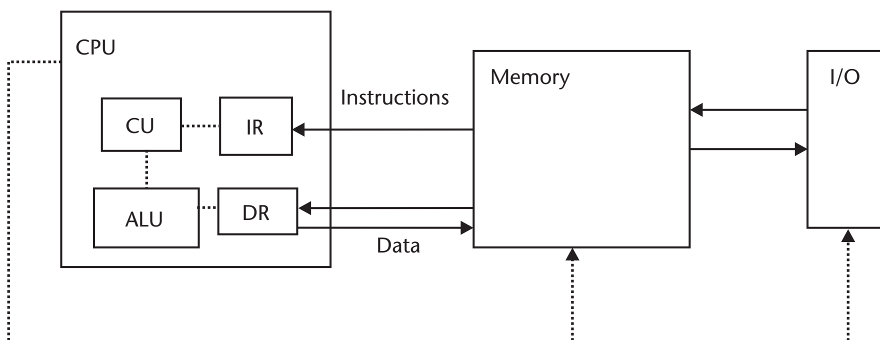

Figure 1.1 Von Neumann model.

The speed of such a machine depends on the time required to load the instruction and to execute it. To increase the computing performance, the von Neumann model has been improved in the last decades. This happened thanks to the evolution of microelectronics, which is more and more able to concentrate chips in small spaces. The availability of more hardware components, in fact, led to designing computer architectures with higher performances than the traditional von Neumann machine. Parallelism was introduced by adding new ALUs (see Figure 1.2) controlled by a common CU or by introducing several cooperating CPUs (see Figure 1.3). A local high-speed interconnection network allowed these components to exchange data.

According to Flynn's classification [2], the former architecture corresponds to the so-called SIMD model, where the same instruction is executed in parallel on different data by the ALUs. According to the SIMD model, the CU broadcasts a single instruction to all of the ALUs, which execute the instruction synchronously on local data.

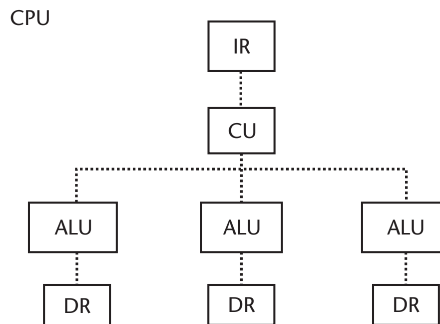

Figure 1.2 Simple instruction multiple data (SIMD) parallel architecture. A single CU controls a number of ALUs. ALUs execute in parallel the same instruction (stored in the CU local register known as IR) on different local data (each stored in a different local register, called DR).

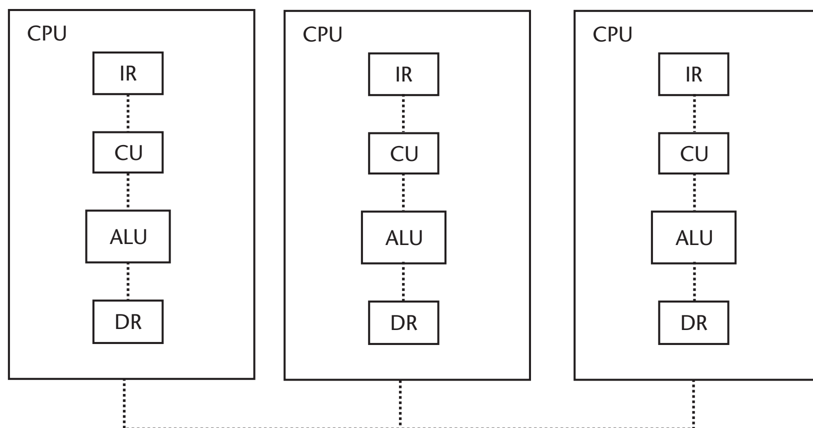

Figure 1.3 Multiple instruction multiple data (MIMD) model. A parallel machine in the MIMD architecture contains a number of CPUs interconnected by a high-speed network. The different CPUs execute in parallel different instructions and operate on different data. In order to achieve a common goal, the processors must synchronize and exchange data.

The latter corresponds to the MIMD model, where the processors interpret in a parallel fashion different instructions, each operating on different local data. Therefore, MIMD computers support parallel solutions that require processors to operate in a largely autonomous manner. They are substantially composed of asynchronous computers, characterized by decentralized hardware control.

MIMD architectures may differ depending on whether memory is shared. The processors, in fact, can address a global, shared memory (in the so-called *shared-memory architectures*) or can each address a local memory (in the so-called *distributed-memory architectures*). The two different models imply different kinds of communication between processors. In distributed-memory architectures, processors share data by explicitly passing messages through the interconnection network, with performances depending on the bandwidth of the network (*message passing* programming paradigm). In the shared-memory architectures, processors must synchronize their access to shared data to prevent one process from accessing one datum before another finishes updating it. In Section 1.4, more details regarding the two architectures are provided, with a particular emphasis on the implied programming paradigm.

In the past, parallel architectures were implemented in the so-called massively parallel processors (MPPs), computers containing hundreds or thousands of processors interconnected by a fast local interconnection network. In recent years, as the price of commodity personal computers has fallen, these special-purpose parallel machines have, for the most part, ceased to be manufactured. Parallel architectures, in fact, can nowadays be implemented as well by connecting a number of isolated machines and by building *clusters*. With the power and low prices of today's off-the-shelf PCs, the availability of networking equipment, and low-cost, mostly public-domain operating systems and other software, it makes sense for a large number of applications to build HPC environments by assembling commodity PCs to form a cluster. Nonetheless, traditional supercomputers are still used for applications with very stringent performance requirements, as well as in several applications that are not strictly amenable for a successful implementation on clusters (e.g., due to fine computational granularity or intensive data communication).

As a result of this trend from built-in supercomputers to the aggregation of machines, nowadays *parallel* applications come along with *distributed* applications, a more general concept defining applications made up of a number of dispersed components performing different tasks that have the capability to interact in order to perform a well-specified goal. Distributed applications run on distributed memory architectures obtained by assembling dispersed computers, often with heterogeneous platforms and operating systems.

Incidentally, the use of such architectures and applications goes along with the spreading of adequate software technologies [3, 4], most of them based on object-oriented (OO) concepts, permitting:

- The assembly of heterogeneous computers;
- The distribution of application tasks;
- A transparent usage of distributed machines, which are perceived as forming a single entity.

Distributed computing technologies work well in local area networks (LANs) or when gluing machines belonging to the same organization. This limitation is currently overridden by the introduction of grid computing.

1.3 Parallel and Distributed Topologies

Distributed memory architectures are implemented in MPPs by connecting nodes with a fast interconnection network. Nodes share data by exchanging messages through the interconnection network. The way nodes are interconnected (i.e., the way the topology nodes form via the interconnection network) must be designed with care, as at least two critical features depend on it:

- The *scalability* (i.e., the easy accommodation of an increasing number of processors);
- The *adaptability* to the requirements of scientific programs (i.e., the ability to minimize communication times given the nature of the problem to be solved and its communication patterns).

A number of topologies have been proposed [5], each being the most appropriate for a different class of problems and situations. An interesting parameter used to classify topologies is the *communication diameter* (i.e., the maximum number of nodes a packet must traverse when traveling from the sender to the destination). What follows is a list of the basic topologies.

1. *Ring*. The N nodes are interconnected to form a ring (see Figure 1.4). Each node is directly connected to two neighbors. The communication diameter is $N/2$ and can be reduced by adding chordal connections. Ring topologies are appropriate for a reduced number of processors executing algorithms with little data communications.
2. *Mesh*. The $N = n^2$ processors are interconnected to form a two-dimensional mesh (see Figure 1.5). Each internal node directly communicates with four neighbors. The communication diameter is equal to $2*(n - 1)$ and can be reduced if wraparound connections at the edges are added. The similarity between this topology and matrix data structures make this topology amenable for matrix-oriented algorithms.

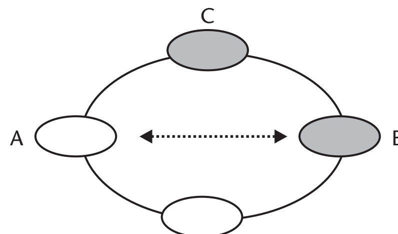

Figure 1.4 Ring topology. The N (4 in the example) nodes form a ring. Each node has two neighbors. The communication diameter is equal to $N/2$ (2 in the example). When node A needs to communicate with node B, the packet must traverse two nodes (C and B).

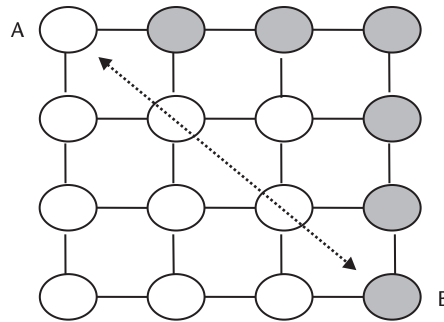

Figure 1.5 Mesh topology. The $N = n^2$ (16 in the example) nodes form a mesh. The communication diameter is equal to $2*(n - 1)$ (6 in the example). When node A needs to communicate with node B, the packet must traverse the six colored nodes.

3. *Tree*. The nodes are interconnected to form a tree (see Figure 1.6). The most diffused tree topology is the binary one and fits well for tree-oriented algorithms, such as searching, sorting, and image-processing algorithms. If the binary tree has got n levels (with $2^n - 1$ processors), the communication diameter is $2*n$ and can be reduced by adding a direct link between nodes at the same tree level.

Recently, a number of software utilities have been developed to allow the user to choose among a set of *logical* topologies (i.e., topologies not necessarily implemented in the underlying physical topology). The software is responsible for efficiently mapping the logical topology to the underlying physical topology (see Figure 1.7).

The growing diffusion of such software tools is itself a demonstration that the perfect topology, ideally suited for any kind of applications, does not exist: the choice of a favorite topology cannot be separated from the application to be run. Consequently, the flexibility of the topology, as well as the ability to dynamically adapt the main networking features of the computational environment, is highly recommended. Flexibility is required more and more when considering a target environment for the immediate future, and this could be an appealing characteristic grids let us foresee. An apparent evidence of the intrinsic link between application

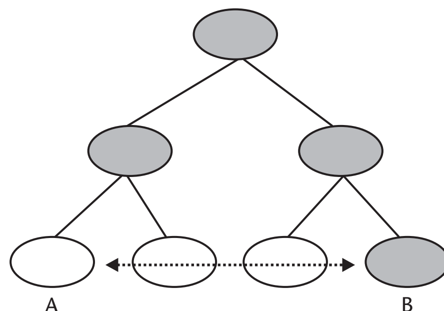

Figure 1.6 Tree topology. The $N = 2^n - 1$ (7 in the example) nodes form a complete binary tree with n levels ($n = 3$ in the figure). The communication diameter is equal to $2*(n - 1)$ (4 in the example). When node A needs to communicate with node B, the packet must traverse the 4 colored nodes.

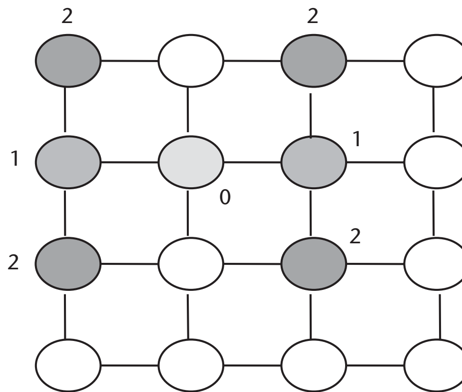

Figure 1.7 Binary tree mapped to a physical mesh. A two-level tree topology is logically mapped onto a reconfigurable mesh. The node labeled with the 0 label is the root of the tree. The nodes labeled with the 1 label belong to the first level. The nodes labeled with the 2 label belong to the second level of the logical tree.

and architectural customization is represented by *adaptive mesh refinement* [6], performed by applications that dynamically adapt their behavior to the availability of resources inside a grid.

1.4 Parallel and Distributed Programming

When a program is run in parallel, program units must have the ability to communicate with one another in order to cooperatively complete a task. As overviewed in Section 1.2, shared-memory architectures allow processor communication through variables stored in a shared address space, while distributed-memory architectures are built by connecting each component with a communication network.

With a shared-memory multiprocessor, different processors can access the same variables. This makes referencing data stored in memory similar to traditional single-processor programs, but adds the problem of shared data integrity. A distributed-memory system introduces a different problem: how to distribute a computational task to multiple processors with distinct memory spaces and gather the results from each processor into one solution.

These differences produce two parallel programming paradigms:

- *Message passing*. In message passing, any interaction among processes is achieved through an explicit exchange of messages.
- *Shared memory*. All program units access data from a central memory and, at any moment, the data can be accessed and eventually changed by any processor node. Every interaction among processor nodes is performed through the shared memory.

Though the architectural difference between distributed- and shared-memory systems is apparent, and it seems quite natural to adopt a message-passing paradigm when using distributed systems as well as a dedicated shared-memory paradigm for the corresponding architecture, in several real cases the addressed application

may request to break the rule. A very trivial example is represented by applications that intensively resort to vector operations, highly amenable to an efficient implementation by using shared-memory facilities. In such a case, even when a distributed memory platform is available, a shared-memory programming paradigm can be very attractive. Indeed, several hardware and software vendors propose dedicated libraries for simulating—most of the times via virtual memory mapping—a shared-memory behavior in a distributed system (an outstanding example was represented by the SHMEM library for Cray T3D and T3E platforms).

This is evidence of the difficulty in proposing a rigid classification of architectural and programming solutions: indeed, the core of the choice is *the application* and the consequent identification of suitable hardware and software strategies.

Being aware of the evanescent separability between programming strategies, hardware solutions, and applications, we propose some details for the two mentioned programming paradigms in Sections 1.4.1 (message passing) and 1.4.2 (shared memory).

Before going on with the description of these programming paradigms, it is worthwhile to recall that a basic form of parallelism can be obtained in multiprocessing environments when using *threads* [7]. A thread can be defined as a stream of instructions that can be scheduled to run as if it were a process with an autonomous identity with respect to the program of which it is part. This means that a thread can run asynchronously with respect to the process that created it, the *parent* process (see Appendix A for an introduction to UNIX processes). Indeed, threads are strictly bound to their parent, as they share critical resources, such as files and memory data, with it (and with any other threads created by it).

To the software developer, the concept of a "procedure" that runs independently from its main program may best describe a thread. A *multithreaded application* is an application in which a number of tasks are carried out in parallel by simultaneously running threads. Multithreaded applications provide good performance in many situations, such as:

- *CPU computation overlapping with I/O*. The program may invoke two threads: one thread waiting for the I/O completion, the other thread making computations.
- *Asynchronous event handling*. Threads may be used to interleave tasks serving events of unpredictable frequency and duration. For example, a Web server can use threads to both transfer data from previous requests and manage the arrival of new requests.

Threads are a simple and effective way of exploiting program natural parallelism and concurrency on multiprocessor hardware.

1.4.1 Message Passing

Existing message-passing libraries are based on two separate standards, parallel virtual machine (PVM) and message-passing interface (MPI). PVM [8], written at Oak Ridge National Lab, is a portable heterogeneous message-passing system. It provides tools for interprocess communication, process spawning, and execution on

multiple architectures. The PVM standard is well defined, and PVM has been a standard tool for parallel computing for several years.

MPI [9] has come into the mainstream more recently than PVM, but it is now a mature standard. MPI has been defined by a committee of vendors, government labs, and universities. The implementation of the standard is usually left up to the designers of the systems on which MPI runs. Anyway, a public domain implementation, MPICH [10], is available.

MPI is designed primarily to support the single program multiple data (SPMD) model, even though it works fine for other models as well. In the SPMD programming paradigm, all tasks execute the same program on different sets of data. All of the nodes receive identical copies of the program to be executed. However, the program can contain conditional statements that execute different portions of the program, depending on the node where the program is executing, thereby enabling the programmer to run different instructions within different tasks.

When an MPI program starts, it spawns a number of processes as specified by the user. Each process runs and communicates with other instances of the program, possibly running on the same processor or different processors. Basic communication consists of sending and receiving data from one process to another.

In the simplest MPI programs, a master process sends off work to worker processes. Those processes receive the data, perform tasks on it, and send the results back to the master process, which combines the results. This is called the *master-worker* or *host-node* model (see Figure 1.8).

More information about MPI and MPICH is in Chapter 2, where a schematic classification of MPI routines is reported together with an introduction to MPICH and MPICH-G2, the MPICH implementation for grid computing, and in Chapter 3, where practical details about how migrating MPI applications to MPICH-G2 are reported. In Chapter 4, practical hints are given with reference to a real application (FDTD implemented with MPI and MPICH-G2).

1.4.2 Shared-Memory Programming

OpenMP [11] is an open standard for providing parallelization mechanisms on shared-memory multiprocessors. Specifications exist for C/C++ and FORTRAN.

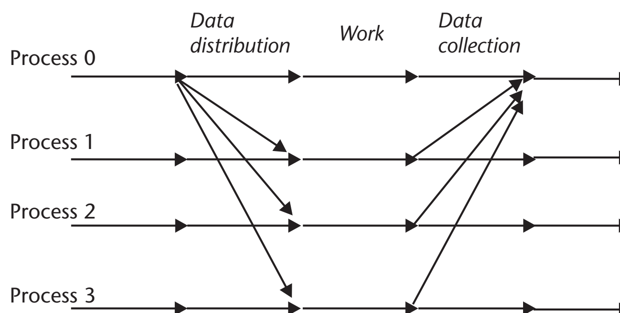

Figure 1.8 MPI execution model. MPI fits well with the SPMD programming paradigm. According to it, programmers write a single program that gives place to multiple processes. Processes exchange data by using MPI library calls. Normally, a root process (process 0 in the example) is responsible for distributing problem data among the remaining processes and for collecting results at the end of the executions.

The standard provides a specification of compiler directives, library routines, and environment variables that control the parallelization and runtime characteristics of a program. Like MPI, OpenMP is portable to other platforms. The compiler directives defined by OpenMP tell a compiler which regions of code should be parallelized and define specific options for parallelization. In addition, some precompiler tools exist which can automatically convert serial programs into parallel programs by inserting compiler directives in appropriate places, making the parallelization of a program even easier. One example is the Visual KAP for OpenMP [12].

1.4.3 Concluding Remarks: Programming Paradigms and Parallel Architectures

Both shared-memory and message-passing paradigms have advantages and disadvantages in terms of ease of programming. Porting a serial program to a shared-memory system can often be a simple matter of adding loop-level parallelism, but one must be aware of synchronization problems related to simultaneous access to the same data.

Writing a message-passing program, on the other hand, involves the additional problem of how to divide the domain of a task among processes with separate memory spaces. Coordinating processes with communication routines can be quite a challenge.

Despite other interesting features, shared-memory systems in general have poor scalability. Adding processors to a shared-memory multiprocessor increases the bus traffic on the system, slowing down memory access time and delaying program execution. Distributed-memory multiprocessors, however, have the advantage that each processor has a separate bus with access to its own memory. Because of this, they are much more scalable. In addition, it is possible to build large, inexpensive distributed systems by using commodity systems connected via a network. However, latency of the network connecting the individual processors is an issue, so efficient communication schemes must be devised.

1.5 Performance Assessment

It is of the utmost importance in parallel computing to assess the speed gain obtained from the operation of N processors in parallel. For this purpose, a parameter called speed-up ratio is introduced. Suppose you run a program using a single processor, and it takes time T(1). Suppose the program is written to take advantage of the available number of processors. You then run the program using N processors and it takes time T(N). Then, we call *computational speed up* [13] the ratio:

$$S = T(1)/T(N) \tag{1.1}$$

and *efficiency* the ratio:

$$E = S/N \tag{1.2}$$

The longer processors are idle or carry out calculation due to the parallelization, the smaller E becomes.

Now, suppose we isolate in the program the strictly serial part (i.e., the part that cannot be parallelized) from the parallel part. Call B the percentage of the strictly serial portion of the program ($B<=1$). Then, the strictly serial part of the program is performed in $B^*T(1)$ time. The remaining part, $(1 - B)$, is demanded to a set of N processors. If we assume that each processor requires $1/N$ time of one processor working alone, then the strictly parallel part is performed in $((1-B)^*T(1)) / N$ time. With some manipulations, we get the formula:

$$S = \frac{N}{(B^* N) + (1 - B)}$$

This is Amdahl's Law [14], which establishes limits to the speed up obtainable when increasing the number of processors. To understand the Amdahl's law, we refer to a speed-up curve. A speed-up curve is a graph with the number of processors on the x axis, and the speed up S on the y axis. The best speed is when $B = 0$ (i.e., the whole program is parallelizable). This would yield a 45-deg curve (i.e., $S = N$). When B is constant, Amdahl's Law yields a speed-up curve that is logarithmic and remains below the line $S = N$. This law shows that it is indeed the algorithm and *not* the number of processors that limits the speed up. Also note that as the curve begins to flatten out, efficiency is drastically reduced.

Amdhal's law is useful to assess the amenability of an algorithm to be parallelized, but it is not a practical tool. In real applications, performance depends on a number of key factors, such as network bandwidth in the case of distributed memory architectures and processor load in multiuser environments. This becomes truer and truer when migrating from dedicated parallel machines to wide area environments, where distributed architectures assemble heterogeneous systems shared among a number of users. To exploit to the utmost the available computing power, processes must be dispatched to the processors by taking into account the current status of resources and their match with job requirements. A number of tools targeted to this job scheduling work exist. Among them we recall Platform's Load Sharing Facility (LSF) [15], and Altair's Portable Batch System [16] and Condor [17]. Job scheduling tools accept, as input, files listing the jobs to be submitted and their requirements. Once job requirements are known, they inspect the status of the connected machines and schedule the tasks by following some load-balancing criteria. In this way users can submit their jobs and later contact the tool to query their status. Some tools provide checkpointing and restart facilities so that computations can be migrated from overloaded or failed machines to lightly loaded ones. The set of machines that are configured to be managed under the same scheduling system is usually called *pool* and is normally administered by a single organization, as the configuration of the pool is at its best when centrally managed.

1.6 Web Computing

The Web was born as a uniform and easy way to *distribute information*. Documents were written in the Web language, the hypertext markup language (HTML), to permit hyperlinks and multimedial representation of data. Servers hosting the same documents were equipped to retrieve the suited HTML file on demand. This simple

architecture follows the traditional *client-server* model (also referred to as *two-tier* model) [18] (see Figure 1.9), in which the *client* is equipped with a graphic interface (browser) from which users can issue their requests, and the *server* is equipped with the software needed to respond to requests coming from clients and hosts the HTML files to be sent when requested.

This is a *static* model, where users can only access and read documents stored in the Web servers.

In the last decade, the Web evolved from the *document publishing* arena to become a *data sharing and computing* environment. This has happened thanks to the development of software technologies that, fitting well with Web models, give the Web the ability to process data, both on the client side and on the server side. The first software technology was the common gateway interface (CGI), which allows a Web server to provide HTML pages generated on the fly on the basis of data stored in an electronic archive. The Java revolution gave a further impulse to this transformation. The Java language was created to develop applications running on the Web and exploitable by the classical Web client tool, the browser. The "write once run everywhere" Java property has the ability to automatically migrate Java applications, located in Web servers, to the Java-enabled client. On the server side, the Java servlets allow enriching Web servers with potentially unlimited computing capability.

The new software technologies make the Web an *interactive* tool, with enriched capabilities:

- Users can query, modify, and mine data disseminated on the Internet;
- Applications can migrate from platform to platform;
- Users can insert their data and manipulate them via the Web.

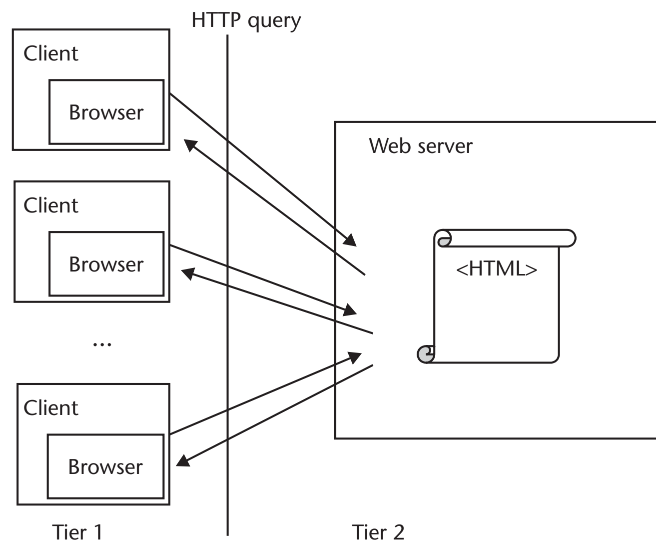

Figure 1.9 Client-server Web architecture. This is the document publishing model. This model contains two tiers—a client tier, where end users make queries by using a browser, and a server tier. Queries follow the hypertext transfer protocol (HTTP). The server answers the HTTP queries by sending back pages written in the HTML language.

In the new scenario, where servers play an active role, application software is separated from data, thus improving the manageability and flexibility of the Web architecture. This gives place to the *multitier architecture* (see Figure 1.10), where a number of machines (three or more) cooperate to build the Web page.

In its simplest realization, this architecture involves three tiers:

- The *front-end* tier, the client from which the end user expresses requests by using a browser;
- The *middle* tier, the Web server equipped with software capable of building HTML pages on the fly, starting from data originated by users as well as data fetched from the backend tier;
- The *backend* tier, a backend database where data are stored.

Now, the Web resembles more and more a giant computer, where users can run remote applications via a uniform interface. Users contact Web servers to perform calculations, mine data, and launch applications. Why not further fragment Web applications in the naturally distributed Internet framework? Once the original, static client-server architecture has been abandoned, why not making the HTML pages originate from the cooperation between distributed application components?

This appears as an easy step: it is sufficient to merge Web technologies and protocols with distributed and parallel technologies. As a result, distributed computing technologies [3, 4], being originally targeted to local area distributed environments, are now opening to Web standards and protocols, thus contributing to this evolution. Also Java produces a component model oriented to the Web environment, called Javabeans [19], which requires that distributed applications are written in the Java language. Particularly interesting are the *Java mobile agents* (JMAs) as well. A JMA [20] is a Java program with the ability to transfer itself from host to host within a network and to interact with other agents in order to perform its task.

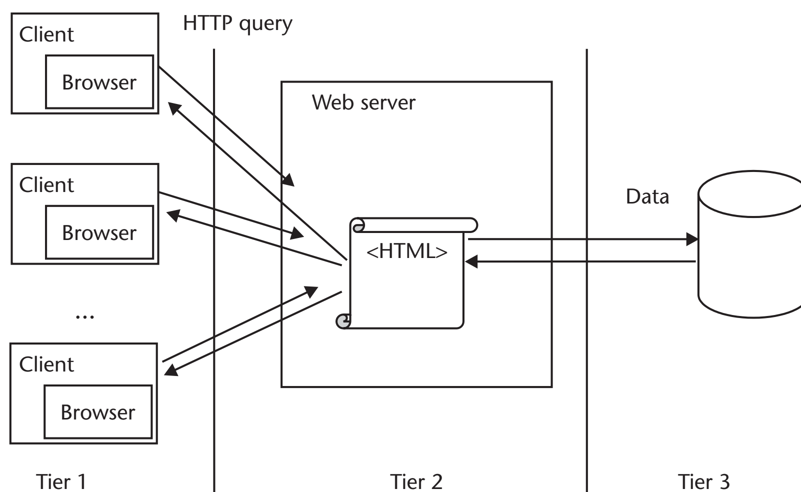

Figure 1.10 Three-tier Web model. The front-end tier is where end users make queries through a browser. Web servers (the second tier, commonly called the application tier) are equipped with software capable of building HTML pages on the fly, starting from data originated by users as well as data fetched from databases, belonging to the third tier.

When an agent travels, it transports its state and code, so that it can have an intelligent behavior and decide autonomously its itinerary and the way it interacts with other agents and objects.

Together with the evolution of distributed architectures towards the Web, a new concept begins to spread among Web users. *Peer-to-peer* computing [21] allows dispersed and networked computers to talk and cooperate without hierarchies in their behavior. In these architectures, each node may behave interchangeably as client or server, depending on the situations. The traditional distinction between server and client machines is vanishing. An example is the SETI@home project [22]: using free software downloaded over Internet, home computers lose their passive role as client and actively contribute with peers disseminated throughout the Internet to the research of extraterrestrial signals.

Such a *componentized* Web can still be improved indeed, migrating distributed concepts from LANs to Internet environments opens a number of new problems. Wide area distributed applications must deal with transient, slow, and unreliable Internet connections. Furthermore, Internet applications must glue components coming from different organizations, each with its own policies and technologies regarding security, scheduling, and so on. Distributed computing technologies do not seem to cope completely with these aspects, being often proprietary or bound to a specific platform or language. On the other side, peer-to-peer computing has given rise to several application-oriented systems without defining a common and general infrastructure.

As a matter of fact, a new technology is needed to take full advantage of the new trends in Web computing. These demands can be fulfilled by *grid computing* technology, which is leading the Web to the even wider transformation towards a *resource-sharing* architecture (see Figure 1.11). A resource-sharing architecture enables diverse resources, including not only software components, but hardware

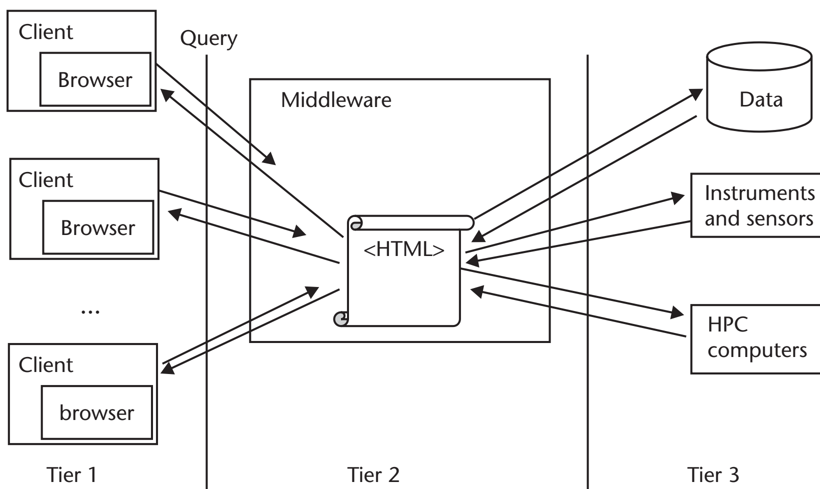

Figure 1.11 Resource-sharing Web. When extending Web functionalities to the sharing of resources (both software and hardware), the middle tier becomes a more general middleware bag of services, including brokering of distributed resources, security services, and application integration. The back-end tier may include high-performance computers (MPP or clusters), tools, data, and scientific instruments. The end users still access backend services by using the browser.

resources and logical entities (such as single domain clusters), to be merged to achieve specific goals.

1.7 Computational Grids

1.7.1 Introduction

As discussed in the previous sections, parallel computing, originally conceived as the exploitation of dedicated and expensive multiprocessor or vector architectures, has progressively switched towards the adoption of distributed paradigms, opening appealing frontiers to low-cost high-performance computing. Meanwhile, Web technologies have emerged, thus enforcing the trend towards distributed applications.

The Web on one side and the distributed computing on the other have remained, until some years ago, substantially disjointed, except for very trivial issues regarding networking. Recently, the technology of computational grids is performing a revolutionary action, joining together these two worlds so that they not only collapse into one concept, but mutually reinforce themselves thanks to a wide standardization effort. This effort induces, as a relevant side effect, the opening of complex and powerful tools to very large numbers of end users, with a consequent increase in the kinds and numbers of possible applications, multidisciplinary actions, and cooperative initiatives. One quite immediate witness of this *digital unification and equalization* is the nearly unlimited availability of computational power to the generic end user inside a large grid, which makes effective HPC affordable for anybody.

Of course, many problems are still barriers to completely achieving the goal (one of the most critical being the availability of high-speed networks). Nonetheless, it is definitely worth exploring such new working methodologies and tools, which will change the management of research projects and facilities as well.

We propose in the remaining part of this chapter a simple description of what a computational grid is, discuss the global architecture and its core, and finally review the nature of several possible applications.

1.7.2 What Is a Grid?

A computational grid [23] consists of distributed software and hardware facilities, such as computational resources, storage systems, catalogs, network resources, and sensors. A resource may also be a logical entity, such as a distributed file system or a computing cluster. GC software pools disperse resources into a unique virtual system and allow anyone on the network to access it (or its facilities). As shown in Figure 1.12, a grid can span domains of different dimensions, starting from local grids made up of nodes connected by LANs, up to global grids, made up of heterogeneous nodes owned by different organizations and connected by the Internet.

As seen in the previous section, GC arises from the emerging need to transform the Web into a giant repository, where users can pick up resources as needed. This is possible because a grid is conceived as a set of universally recognized standards, protocols, and Web-compliant technologies open to the majority of existing distributed visions and methodologies.

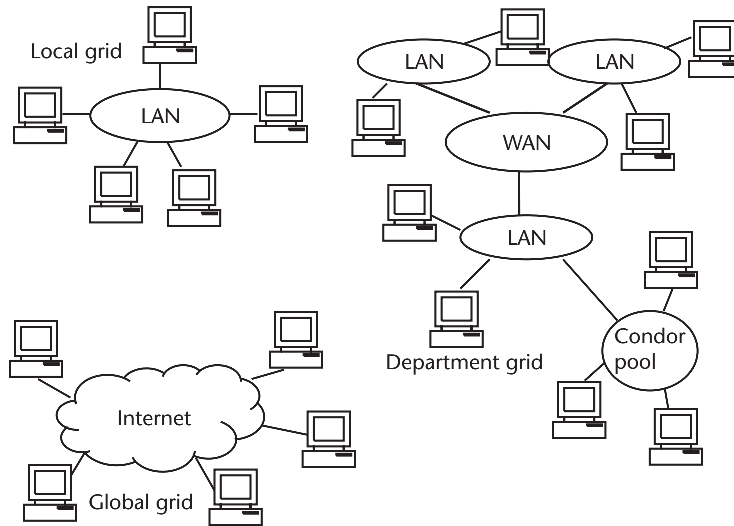

Figure 1.12 Examples of grids. A LAN can host a local grid, and its local grid can itself be part of a wider grid—for example, at wide area network (WAN) level. A distributed system (Condor pool in this figure) can in turn be another portion of the grid.

GC technology:

- Talks with Web standards and protocols;
- Integrates established software technologies for Web and distributed computing.

Rather than summing these two features, grids envelope them in a larger and pervasive environment. To do this, grids cope with the challenges related to the Internet environment:

1. *Fault tolerance*. Grids are complex environments, including a huge number of heterogeneous entities, each of which may fail at any moment. Robustness with respect to failure of network connections, machines, software components, and so on is then a critical issue.
2. *Security*. The Internet is intrinsically insecure and decentralized. When defining a grid, users must be recognizable and access to resources must be traced and controlled.
3. *Dynamism*. The Internet environment is continuously changing, resources may be added or removed at any moment, and their status (load, traffic, and so on) is variable. Grids must tailor their behavior in agreement with changing conditions of the environment.
4. *Scalability*. Once operative, a grid is presumed to increase its number of resources and users. Grids performance must not be affected by this.
5. *Heterogeneity*. Grids resources are heterogeneous: network, platforms, operating systems, electronic devices, and software tools provided by different vendors and following different architectures and paradigms are merged in a grid. Grids must define uniform and standard ways of interaction with them so that heterogeneity is hidden.

6. *Autonomy*. Grid resources belong to diverse organizations. Grids must federate these resources by leaving owners free to establish and implement their own policy regarding security, scheduling, and so on.

Meanwhile, other requirements are fulfilled to simplify the interaction with the end user:

I. *Transparency*. Users must access the dispersed resources while perceiving them as a whole. Location and access to a resource must be straightforward, both if the resource is local and if it is remote.
II. *Uniformity*. The interaction with a grid must happen via a uniform interface, possibly the Web browser.
III. *Homogeneity*. Grids must mask to end users their underlying heterogeneity, allowing the access to each resource without taking care of its peculiar characteristics.

As discussed so far, grids must manage with a multiplicity of resources, with the main goal of guaranteeing a simple access to them. This requires dedicated software interfaces to drive resources, as well as suitable software tools to allow a user-friendly interaction with such drivers. More schematically, grids can be seen as composed of three kinds of entities:

- Resources;
- Grid software hiding the complexity of the Internet environment and satisfying requirements 1–6;
- Tools for the interaction of end users with the grid and fulfilling requirements I–III.

These three groups result in a three-level architecture: *fabric level, middleware level,* and *application level*. The intermediate layer, also called *grid middleware* (GM), contains the core grid software. The GM mediates between the other layers, talking with them via well-defined protocols and application programming interfaces (APIs), so that integration of heterogeneous technologies and encapsulation of legacy systems are possible. End users work with the grid by using grid-enabled applications and software tools (at the application level). Grid-enabled applications and software tools operate with grid resources (belonging to the fabric level) via a number of *services* provided by the GM layer.

In the following section, a schematic view of the grid architecture is given, while in Section 1.7.4 the most fundamental grid services offered by GM are summarized.

1.7.3 Grid Architecture

As introduced earlier, the grid architecture [24] contains three layers (see Figure 1.13):

- *Fabric level,* which includes everything the grid must glue. They are all of the resources belonging to the grid, dispersed in the world and interconnected by

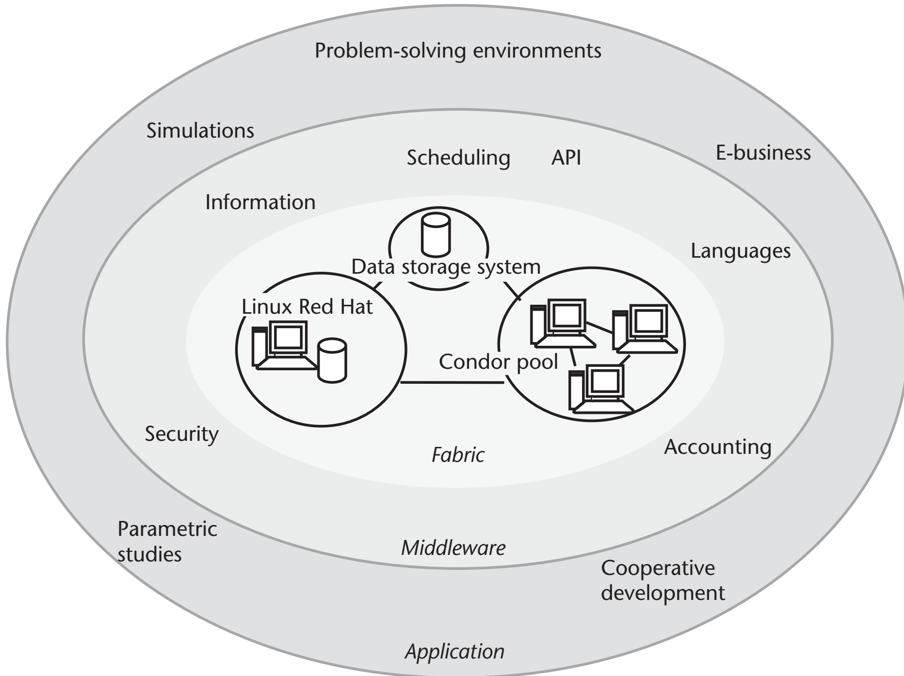

Figure 1.13 Grid environments have a layered architecture. On the top of the architecture there is the application level, where users develop and use grid-enabled applications. Applications are implemented with development tools (e.g., APIs, libraries, and languages) provided by the middleware level and operate in a distributed environment by using such services as information management (IM), allocation, and scheduling provided by that level. The middleware level hides the complexity of grid environments, made up of a number of heterogeneous resources, forming the fabric level. The fabric level consists of network resources (devices and communication protocols), computing software resources (local resource management tools, operating systems, and data storage systems), computing hardware resources (CPUs and storage devices), and scientific instruments.

the Internet. Resources can be *physical*, such as hardware (CPU, memory, electronic devices, network) and software (application components, databases) entities, or *logical* (clusters, distributed pools). All of the packages providing basic services used in local domains also belong to the fabric layer. For example, local resource managers, such as operating systems and distributed management systems [the already mentioned LSF, Condor, and Portable Batch System (PBS)], as well as security tools (such as Kerberos), are included in this layer. The fabric level composition is dynamic, as the set of resources can change over time. Resources are shared among grid users, whose number changes over time as well.

- *Middleware level*, which includes the software responsible for mediating between the resources and their higher level *managers* in order to hide to grid end users and application developers the complexity of the fabric level. The GM operates on grid resources and the local managers (i.e., single domain schedulers, allocators, and load balancers) to offer core grid *services* to distributed applications. This level contains basic elements needed to develop grid-enabled applications as well. This means that it also contains libraries and languages oriented to grid development.

- *Application level*, which is the level with which end users interact. It includes both high-level services that allow software developers to implement grid-aware applications and Web tools to permit end users to work with the grid by submitting jobs, collecting and analyzing results, and cooperating with remote colleagues. Any grid-oriented application belongs to this level as well.

1.7.4 Grid Middleware

Grid resources are supposed to be geographically distributed and owned by different organizations, each with proprietary policies regarding security, resource allocation, platform maintenance, and so on. Such an environment strongly depends on the construction of a robust infrastructure of fundamental services able to smooth out mismatches among different machines, security and scheduling policies, operating systems, platforms, file systems, and so on. The middleware layer is in charge of playing this role. Example of services provided by GM are:

- *Allocation of resources*. Access to resources, CPU time, memory, network bandwidth, storage systems, and so on, has to be carefully scheduled in order to extract the maximum performance from them. For example, users must be given the ability to schedule their jobs, to track their behavior, and to analyze the status of allocated resources. Application components must be able to change their working machines to improve load balancing or because of a failure. A number of resource managers [15–17] exist that work well in single-domain distributed environments. When including such systems, grids provide a uniform and transparent interface to them. This means that when a user connects to a grid, he allocates grid resources by simply specifying the application requirements. The grid itself is responsible for dispatching requests to resources, eventually contacting the local resource managers.
- *Computational economy*. Grids federate resources coming from heterogeneous organizations. Some organizations could decide to rent their resources upon a payment. Accountability services, accompanied by an ad-hoc economic model, give providers of grid resources such an opportunity. In this way, consumers pay for their use of servers, storage, CPU, and so on. These resources are metered through software measurement tools and can be billed to consumers. In Chapter 2, billing services are explained when introducing grid architecture for computational economy (GRACE), an architecture designed for integrating resource managers with economic models, so that resources can be chosen on the basis of their price as well.
- *Information management*. This allows the continuous monitoring of resources and their status. The information management service implements two mechanisms, *registration* and *discovery*. Registration is in charge of allowing entities to enroll themselves as part of the resource pool and communicate their characteristics to the whole grid. Discovery locates and accesses the resources and their attributes once they have been registered. Information management service relies on the creation of a universal naming service integrated with existing protocols and with established conventions for accessing resources, such as file transfer protocol (ftp) and HTTP.

- *Security*. Resource sharing must be highly controlled, with resource providers and consumers clearly defining what is shared, who is allowed to share, and the conditions under which sharing occurs. This implies that the grid must allow resource owners to define authorization policies to moderate the access to their resources and that some services must exist monitoring who accesses resources and when. Furthermore, authentication and, if required, confidentiality must be granted (see Appendix B for an introduction to these concepts). A number of tools exist that guarantee security services at the single domain level, such as Kerberos [25]. Grids must also in this case interact with lower level services to provide a transparent access to them via a uniform, high-level service.
- *Data management*. Data management is focused on access and transfer of data. When large amounts of data are managed, speed, security, and reliability become key factors. A number of distributed storage systems such as High-Performance Storage System (HPSS) [26] from IBM, Distributed Parallel Storage System (DPSS) [27] from LBNL, and Storage Resource Broker (SRB) [28] from San Diego Supercomputer Center actually provide these services. These systems are proprietary or work well in single-domain contexts. Grids provide data-management services in a standardized way, so that previously installed distributed storage systems can talk to one another and be absorbed into a unified system for data-intensive applications. Grids provide services to create, manage, and access data sets replicas as well. Replicas are multiple copies of data stored in different, dispersed sites. When access to these data is needed, the nearest copy is chosen, so that performance is optimized.

In Chapter 2 and in Chapter 3, these services are analyzed into details, with reference to a well-known tool for grids implementation, namely the Globus Toolkit.

1.7.5 Applications

Grid technology opens a new spectrum of applications. Several areas may benefit from the new environment, such as *collaborative engineering*, *data exploration*, *HTC*, the so-called *meta applications,* and of course *HPC*.

Collaborative engineering means providing engineers and researchers with tools for cooperating online. These tools allow them to share remote resources and design, implement, and launch applications in cooperation. In such a context, a grid can be seen as a global production environment, where distributed systems can be prototyped and tested. The flexibility of grids makes this system dynamic and configurable. Via the grid, researchers are able to rapidly modify their products in order to adapt them to the changes of underlying environments, infrastructure, and resources.

Data exploration with grids allows managing and mining distributed data. A number of research fields, such as climate analysis and biological studies, benefit from grid technology as they need storing and accessing data up to the terabyte or petabyte range. Grids distribute data on dispersed sites and allow access to them uniformly. Performance and reliability are improved by replicating data sets on multiple sites. Grids gather data originated by a multiplicity of sources as well. Sensors,

scientific instruments, and devices merge the data they produce into a unique virtual pool, where information can be extracted and analyzed through uniform interfaces.

HTC applications require large amounts of computational power over a long period of time. Examples of HTC applications are large-scale simulations and parameter studies. HTC environments try to optimize the number of jobs they can complete over a long period of time. A grid allows exploiting the idle CPU cycles of connected machines and using them for HTC applications.

*Meta application*s are applications made up of components owned and developed by different organizations and resident on remote nodes. A meta application is usually a multidisciplinary application that combines contributions coming from differently skilled scientific groups. A meta application is dynamic (i.e., it may require a different resource mix depending on the requirements of the current problem to solve). Besides, research teams preserve their properties on their own application components by granting the usage through a recognized brokering entity.

HPC applications require a huge amount of power, usually over a short period of time. This is the primary field for which grids were conceived, as a direct evolution of parallel and distributed concepts. Scientific simulations for weather forecast or astrophysics research are an example of applications requiring huge computational power. Before grid explosion, the scope of these simulations was limited by the available computational power, even when using supercomputers. Nowadays, grids allow scientists belonging to diverse organizations to join their computational resources and acquire amounts of computational power otherwise unaffordable.

All of these applications are exemplified in this book; we suggest reading Chapter 4 for an HPC application, Chapter 5 for an application focused on collaborative engineering and meta applications, and Chapter 6 for an application oriented to data exploration and HTC.

1.8 Conclusions

Grid technologies are the convergence of parallel and distributed computing with several other areas (e.g., Web and security). The GM is the place where the multidisciplinary nature and the complexity of grids are managed and masked to the application developer. In other words, GM is the place where the scientists interested in migrating their applications must more largely invest in order to guarantee themselves an adequate return. Consequently, GM is the core object, according to the goals of this book, and it will be dealt with in the next chapters when GM tools and technologies are examined into details.

References

[1] Hennessy, J., and D. Patterson, *Computer Organization & Design*, San Francisco, CA: Morgan Kaufmann Publishers, 1998.

[2] Flynn, M. J., "Very High Speed Computing Systems," *Proceedings. IEEE*, Vol. 14, 1966, pp. 1901–1909.

[3] Thai, T. L., and Andy Oram, *Learning Dcom*, O'Reilly & Associates, April 1999.

[4] CORBA, http://www.omg.org.
[5] Duncan, R., "A Survey of Parallel Computer Architectures," *IEEE Computer*, Vol. 23, No. 2, February 1990, pp. 5–16.
[6] Allen, G., E. Seidel, and J. Shalf, "Scientific Computing on the Grid," *Byte*, Spring 2002, pp. 24–32.
[7] Butenhof, D. R., *Programming with POSIX Threads*, Reading, MA: Addison-Wesley, 1997, pp. 1–12.
[8] Dongarra, J., et al., "Integrated PVM Framework Supports Heterogeneous Network Computing," *Computers in Physics*, April 1993.
[9] The Message Passing Interface Standard, http://www-unix.mcs.anl.gov/mpi.
[10] MPICH, http://www.mcs.anl.gov/mpi/mpich/download.html.
[11] OpenMP C and C++ Application Program Interface, OpenMP Architecture Review Board, October, 1998, http://www.openmp.org/specs.
[12] Visual KAP for OpenMP, http://www.kai.com/vkomp/_index.html.
[13] Lewis, Ted G., and Hesham El-Rewini, *Introduction to Parallel Computing*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1992, pp. 31–32, 38–39.
[14] Schendel, U. "Introduction to Numerical Methods for Parallel Computers," Ellis Horwood Lim. Publishers, Chichester, UK: 1984.
[15] LSF, http://www.platform.com.
[16] PBS, http://www.altair.de.
[17] Condor, http://www.cs.wisc.edu/condor.
[18] Fox, G. C., "Portals and Frameworks for Web Based Education and Computational Science," http://www.new-npac.org/users/fox/documents/pajavaapril00.
[19] Monson-Haefel, R., *Enterprise JavaBeans*, O'Reilly & Associates, October 2001.
[20] Siniaris, C. G., et al., "Implementing Distributed FDTD Codes with Java Mobile Agents," *IEEE Antennas and Propagation Magazine*, Vol. 44, No. 6, December 2002, pp. 115–119.
[21] Peer to peer working group, http://www.peer-to-peerwg.org.
[22] SETI@home, http://setiathome.ssl.berkeley.net.
[23] Foster, I., C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int. Journal of High Performance Computing Applications*, Vol. 15, No. 3, 2001, pp. 200–222.
[24] Baker, M., R. Buyya and D. Laforenza, "The Grid: International Efforts in Global Computing," *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, Italy, 2000.
[25] Steiner, J., et al., "An Authentication System for Open Network Systems," *Proc. Usenix Conference*, 1988, 191–202.
[26] HPSS, http://www.sdsc.edu/hpss.
[27] DPSS, http://www-didc-lbl.gov/DPSS.
[28] SRB, www.sdsc.edu/DICE.

CHAPTER 2

Enabling Technologies and Dedicated Tools

2.1 Introduction

As discussed in Chapter 1, the implementation of a grid must cope with a number of challenges related to the complexity of the involved environments. Grids, in fact, include a multiplicity of resources that are heterogeneous in nature and might span numerous administrative domains. Each domain may contain its own policies and tools regarding security, scheduling, and resource allocation. Grids should include some software able to glue all of the different installed tools, services, and protocols in order to provide all of the high-level services needed for transparent access to resources as if they were belonging to a single unified "metacomputer." This can be achieved by a layered architecture [1], as the one shown in Figure 1.13, where an intermediate-level software (GM) mediates between the resources and their managers (both belonging to the fabric level) and grid applications (the application level). The GM operates on grid resources and their local managers (i.e., single domain schedulers, allocators, and load balancers) to offer core grid services to distributed applications. Examples of grid services are remote process management, allocation of resources, storage access, information management, and security, which have already been discussed in Chapter 1. Each grid service must be achieved by harmonizing and hiding low-level services and must be exploitable by applications in a straightforward way.

A number of technologies (called *enabling technologies)* help to implement this architecture. These technologies include protocols, programming paradigms, standards, and services, which were introduced before grid evolution and in contexts unrelated to grids. They paved the way for the development of grid concepts and are nowadays the building blocks of GC.

The most outstanding enabling technologies are *security*, which spread out after the boom of Internet, and *object orientation*, which was devised to improve software reusability and portability. Security is discussed both in Appendix B, where the basic principles are introduced, and in this chapter, where security mechanisms as implemented in the most widespread GM tools are overviewed. Object orientation and its relationship with grid technology are described in the following section.

The remaining part of the chapter is devoted to tools supporting grids. In fact, if the enabling technologies paved the way to the grid booming, the real grid evolution took place when the first grid tools were developed. The tools providing basic services (the already-cited remote process management, allocation of resources, storage access, information management, and security services) belong to the GM and form

a software level above which application-oriented tools are built. Therefore, in this chapter, a brief overview of both GM and application-oriented tools is given, with a focus on the GM toolkit called Globus, which is currently the de facto standard adopted in GC.

2.2 Enabling Technologies: Object Orientation

OO concepts originated in the programming-language domain [2], to reduce the cost and complexity of software design, implementation, and maintenance. Since then, several computing disciplines, such as software engineering, databases, artificial intelligence and distributed systems, have adopted the OO principles.

At the moment, OO technologies have two main fields of application: the former deals with *software engineering*; the latter deals with *enabling technologies* and consequently distributed computing. In the framework of this book, attention is prevalently paid to the latter issue, and this section is mostly devoted to a very basic introduction to the concept of OO enabling technologies. Nonetheless, it is worth spending a few lines to give a schematic description of the main concepts of OO software engineering (see also Section 2.2.5 for a more detailed analysis of such an issue in the EM arena).

2.2.1 Object Orientation for Software Engineering

The impact of object orientation on the rethinking of software design and development procedures is evident, especially when recalling that the traditional cycle of production of numerical EM software can be considered as the serial processing of three main steps:

- Problem analysis;
- Code design;
- Implementation.

These steps, in a traditional approach, are each tackled with different tools and models.

In the first step (*problem analysis*), a conceptual model of the problem is formulated and an unambiguous description of it is given with appropriate symbolic schemas (such as flow diagrams). For instance, when dealing with a guided-wave problem in anysotropical media, Maxwell's equations with suitable boundary conditions and constitutive relationships are adopted.

In the second step (*code design*), the design strategy is defined by selecting a certain numerical technique (FEM or FDTD).

Finally, the *implementation step* is performed by identifying a computer language and a programming paradigm.

The long pathway from the first to the final step unavoidably implies a progressive and impressive loss of abstraction and generality. In the code-design step, the general and abstract model of the first step is translated into data structures and functions, then implemented in a specific software language in the third step, with a

relevant loss of information demonstrated by the difficulty of returning to the analysis step once the software has been written and starts to evolve. This results in a difficult scalability and maintenance of software.

On the contrary, in object orientation, the three steps have quite less apparent boundaries. The same object model (the one formulated in the first step) is used throughout the whole process, up to the implementation step. The identification of suitable objects to describe the problem (the first step) is crucial in order to deal with the widest range of applications up to the implementation level, thus ensuring high *adaptability*, *reusability, maintainability,* and *flexibility* of the code. These keywords are the synthetic explanation of object orientation's appeal for software engineers. These themes are thoroughly proposed in the literature. Probably, one of the most complete, critical, and educationally oriented discussions can be found in [3], where the interested reader is referred to for further details.

2.2.2 Object Orientation for Enabling Technologies

OO concepts relevant for enabling technologies can be found in the following features [3]:

- *Encapsulation*. Object orientation represents each entity as *objects* that group data and operations. Each object *hides* its data and exposes a well-defined *interface* (i.e., the operations) allowing operation on hidden data.
- *Abstraction*. Entities (objects) having common properties can be grouped to form a *class*.
- *Polymorphism*. Classes can overlap and intersect (i.e., they can include a common set of operations), eventually assuming different meanings depending on the class to which they are applied. An example of polymorphism is *inheritance*, which allows definition of subclasses of a given class, which inherit class operations.

Objects form a natural model for distributed systems because distributed *components* (objects) can communicate with one another by using messages addressed to their interfaces. The interface to each object is defined very strictly. By contrast, the implementation of an object—its running code and its data—is hidden from the rest of the system (that is, *encapsulated*) behind a boundary the client cannot cross. Clients access objects only through their advertised interface, invoking only those operations that the object exposes through its interface and referring only to those parameters (input and output) that are included in the invocation.

The feature of encapsulation fulfills the heterogeneity and autonomy requirements of large-scale distributed systems. As for heterogeneity requirements, encapsulation means that messages sent to distributed components depend only on the component's interfaces, not on their internals. As concerns autonomy, components can change independently and transparently, provided they maintain their interfaces.

The most recent OO trends are evolving so that the complete set of resources available on a distributed network—including computers, network facilities, data, and programs—can be treated as a commonly accessible collection

of objects. *Distributed object computing* [4] is a computing paradigm that allows objects (*components*) to be distributed across a heterogeneous network and allows each of the components to interoperate as a unified whole.

Distributed object models and tools extend the OO programming paradigm. The objects may be distributed on different computers throughout a network, living outside of the application (*container*) that uses them, and yet appear as though they were local (i.e., resident within the application). Software systems development benefits from distributed object components, as they can be based on prebuilt components, which can be reused and assembled thanks to the features of object orientation.

The movement toward this sort of full object orientation is witnessed by the continuous progress in the specifications for distributed object computing frameworks, the most relevant being overviewed in Table 2.1.

In the following sections, we focus on a couple of outstanding OO frameworks for distributed systems, the CORBA and some of the Java-related systems. The former has been chosen as representative of fully OO distributed frameworks; the latter, because of its suitability for Web-oriented environments. Both often constitute a component of grid environments, interacting with other technologies in wide area networks, as shown in Section 2.12.

2.2.3 CORBA

CORBA was specified by the OMG [7], a company consortium with several hundred member organizations dedicated to producing specifications for OO environments.

A CORBA system consists of an arbitrary number of distributed nodes and clients. Nodes contain application programs and database systems, which constitute the system's computing resources. Clients request operations to be performed by

Table 2.1 Most Relevant Distributed Object Computing Frameworks

Name	*Description*
Open Source Foundation (OSF) Distributed Computing Environment (DCE) [5]	DCE provides a traditional remote procedure call mechanism for the communication between client and server. DCE supports encapsulation as servers accept only the operations defined in their interface.
Microsoft Distributed Component Object Model (DCOM) [6]	DCOM is a Windows-centric platform for distributed computing.
Object Management group (OMG) Common Object Request Broker Architecture (CORBA) [7]	CORBA is a fully OO specification, based on the development of object request brokers able to mediate between clients and server components.
JavaBeans [8]	JavaBeans is the component architecture for Java. The JavaBeans specification allows the transformation of Java classes into beans (i.e., components) by making little changes to the code.
Jini [9]	Jini is a Java-based distributed architecture based on service registering and discovery through an intermediate lookup service.
JMAs [10]	Mobile agents are intelligent components, written in Java, that are able to travel in the network to fulfill their task.

the system resources. One or more distributed object managers, called object request brokers (ORBs), act as interface between clients and resources. Clients do not need to know the location and implementation details of the resources: they connect to the ORBs via well-defined interfaces. ORBs translate the requests so that the target components can interpret them, and they send the results back to the clients.

A language, the interface definition language (IDL), allows object implementations to inform potential clients about the operations they offer and how these should be invoked. In this manner, even legacy applications can be encapsulated into CORBA systems by writing the appropriate IDL wrapper.

OMG's definition of CORBA contains the fundamental guidelines to implement OO distributed systems. OMG leaves ORBs developers the freedom to choose the manner in which and the tools with which ORBs are implemented. This causes some problems of scarce interoperability among different ORBs implementations. Because of this and of its complexity, CORBA is currently quite widespread among single-domain distributed systems but still has difficulty penetrating wider environments. For this reason, most GM tools cope with CORBA deficiencies and have been designed to interoperate with CORBA, thus preserving existing CORBA implementations in local domains and integrating these domains with others to compose a grid.

2.2.4 Java

Java [11] is an OO language whose boom was related to its being the first Web-oriented language, fully developed to fulfill requirements of Web applications. The most relevant feature of Java is portability, thanks to its ability to move code from the server to the client machine. When a Java program is written and compiled, the result is not a processor-specific code, but the so called virtual-machine code. When a client connects to a server to run Java applications, the virtual machine code migrates to the client platform, where it is interpreted by the local browser (see Figure 2.1). Java has built-in security features, which on one side guarantee that malicious code cannot run on client machines, while on the other side constitute a limitation to the language potentialities. Another limitation is speed with respect to processor-specific code: programs written in the Java language cannot be tailored and optimized for the processor where they are going to run (which is unpredictable).

The success of Java has rapidly generated the extension of the language to a number of tools, architectures, and frameworks. A typical context where Java technologies have a competitive advantage is the one related to Web applications or, more generally, computational agents distributed inside heterogeneous networked systems. Java *servlets*, for instance, are a well-known technology enabling the writing of Java pieces of code that cooperate with Java-compliant Web servers to provide services to Web clients. A Java servlet can interface with Web servers with databases and other back-end services and elaborate data to give back the results to the Web.

The experiences with Java servlets have paved the way towards a systematic development of distributed object computing, which is currently supported by several frameworks, two leading examples being Jini and JMAs.

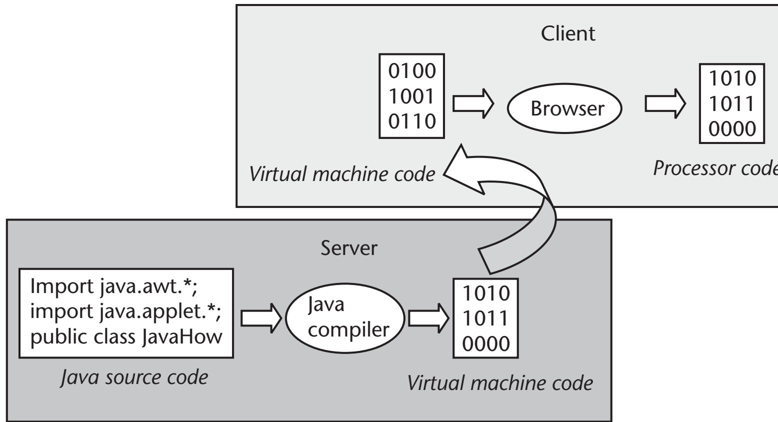

Figure 2.1 Java virtual machine. The Java language allows programs to be written that can be executed on several heterogeneous platforms by code moving and intermediate compiling. The program is written in Java and compiled into an intermediate bytecode, the virtual machine code. When a client connects to the server, the virtual machine code is moved to the client, where a compiler embedded into the browser translates the virtual machine code into the local processor code, making it suitable to run locally.

A Jini system [9] consists of services (i.e., software entities written in Java language that can respond to requests coming from programs or other services). Examples of services include printing a document or returning data about a device. Jini supports a number of specific services that are responsible for the location of distributed services and for calling them. Service requests are satisfied by moving service code from the server to the client with a Java virtual machine procedure.

A *JMA* [10] is a Java program with the ability to transfer itself from host to host within a network and to interact with other agents in order to perform its task. When an agent travels, it transports its state and code, so that it can behave intelligently and decide autonomously its itinerary and the way it interacts with other agents and objects. Thanks to the mobility of agents, JMA systems do not require the compiled application code to be installed on the remote systems. This characteristic distinguishes JMAs from other distributed frameworks (such as MPI, which is discussed in Section 2.10): it allows the design of flexible and dynamic distributed systems (i.e., systems where the computing nodes can be changed dynamically during the execution of the distributed application).

Unfortunately, Jini and other Java-related technologies suffer from the previously enumerated Java limitations, plus their intrinsic constraint of requiring that application components must be written in a specific language. Nonetheless, they are a fundamental component of grid applications when cooperating with other technologies through middleware services.

2.2.5 Object Orientation and Electromagnetic Simulators

The EM community has discovered the interest and charming flavor of object orientation with some delay, with respect to other scientific contexts, such as signal or image processing. As reviewed in a recent paper [12], the first relevant papers in the EM literature were published around mid 1990s.

Despite the challenging perspectives opened by object orientation as an enabling technology (which are directly related to the applications proposed in the following sections of this book), the large majority of the available examples of OO applications to EM problems are based on the implications of object orientation in EM software engineering. The development of EM software can take substantial advantages of OO technologies, and it seems reasonable to predict a continuous increase in the use of such methodologies in EM numerical methods in the forthcoming years.

One of the immediate explanations of this trend is related to the increasing efficiency of numerical techniques, as well as available computing resources. This induces a continuous growth of the complexity of affordable circuits and systems, thus compelling dedicated strategies of domain and problem partitioning, technique hybridization, and system integration [13–15]. This leads to the need to solve different portions of the same problem with different methods, so that the peculiar advantages of a certain approach are exploited where they make the difference, without paying for their drawbacks where they could be more apparent. This casts severe problems of integrating interfaces among heterogeneous methods and codes, as well as the need to solve the problems without affecting the potential independency among the different codes. Consequently, a demand for interoperability, requiring a high capability of information sharing among different methodologies and implementations, cannot be satisfied without preserving the ability of hiding proprietary information.

The characteristics of object orientation, shortly summarized in Section 2.2.1, render this approach extremely suitable to fulfill such requirements. For the sake of brevity, we address the interested reader to [9–20] as reported in [12] for a panel of applications. Other interesting and more recent experiences in the field of computer-aided design (CAD) of MW circuits are also available [16] and even in the development of general-purpose FDTD tools [17].

Unfortunately, as mentioned earlier, EM OO applications related to enabling technologies are rather seldom encountered. One of the few exceptions is represented by the exploitation of OO amenability to code encapsulation and integration. An effective experience was performed in the late 1990s to develop an OO software encapsulator [18, 19] (i.e., a framework devoted to the integration of heterogeneous software tools for antenna design). A more recent exception, very close to the purest concept of enabling technology, is represented by an FDTD implementation based on JMAs [11]. Despite these couple of counterexamples, it is a matter of fact that a substantial limitation persists in the diffusion of OO enabling technologies. GC and its penetration inside the EM community can help to fill in the gap, thanks to the consequent increase of diffused knowledge on the concepts of enabling technologies as well as on their promising perspectives.

2.2.6 Conclusions

In this section, we have overviewed the OO paradigm and its suitability for distributed environments, as well as its suitability to EM applications. The choice of the appropriate enabling technology is not an easy task, as each one has its own potentialities, limits, and drawbacks. None of them must be rejected a priori, as each contains specific benefits for distributed environments.

In a good number of cases, the different technologies are able to talk to one another, via well known protocols and standards. Therefore, the best approach is to choose the set of technologies that appears to be the most suitable for the specific context and to set up an environment allowing them to interoperate. On the other hand, interoperability is viable if a GM software is installed, thus providing basic services for the chosen technologies. Consequently, GM tools are crucial for the effective use of enabling technologies. In the next section, the most widespread technologies for GM are overviewed.

2.3 Dedicated Tools: Grid Middleware

As seen in the introduction of this chapter and in Chapter 1, the implementation of a grid requires the existence of a software layer, the GM, able to provide all of the basic services needed for transparent access to resources as if they were belonging to a single unified metacomputer.

There are many projects worldwide aiming at achieving this ambitious goal. Legion [20] and Globus [21] are the most widespread.

Legion is a middleware developed at the University of Virginia. It embraces the OO paradigm (i.e., it encapsulates all grid components as objects). The methodology used has all of the normal advantages of OO approaches, such as data abstraction, encapsulation, and polymorphism (as discussed in Section 2.2). This approach can be potentially ideal for designing and implementing a complex environment, such as a metacomputer. However, using an OO methodology does not solve a number of problems, the hottest of which is the need to interact with legacy applications and services.

Unlike Legion, which tethers the end user to the OO programming paradigm, the GT offers services and libraries accessible with a dedicated API, from which the developers can choose the ones best fitting with their application. Due to its flexibility and high interoperability with the most common technologies for distributed and parallel computing, GT is rapidly becoming the GM de facto standard and has been chosen for our experimentation. It is described in the following sections.

2.4 The Globus Toolkit: An Overview

GT is a joint initiative of the University of Southern California, the Argonne National Lab, and the University of Chicago. It provides an open-source set of services addressing fundamental grid issues, such as security, information discovery, resource management, data management, and communication.

GT is described by its authors as being made up of three pillars [22]. The first one, *resource management* (RM), allocates resources provided by the grid to the respective consumer. The second one, *information services* (IS), provides information about available resources and their attributes. The third, *data management* (DM), deals with accessing and managing data in a grid (e.g., it provides a more robust and high-performance ftp, customized to grid needs). Each pillar embeds core services given by *Globus Security Infrastructure* (GSI). GSI ensures fundamental security services such as authentication, confidentiality, and integrity.

As GSI is a shared infrastructure among the three pillars, we propose in the next section an overview of this relevant part of GT. Later, we give a detailed description of each pillar (Sections 2.6, 2.7, and 2.8), as well as some important related tools for grid-oriented application development (Sections 2.9 and 2.10).

2.5 The Globus Toolkit: The Globus Security Infrastructure

To introduce GSI protocols [23], we now recall the case depicted in Figure 2.2 (i.e., the access of a generic user to grid services).

In the grid transaction shown in figure, the following steps take place:

- User A logs onto a client machine and asks to start a process Pa on a server machine;
- Process Pa starts on a server machine;
- Process Pa asks to read data resident on a third machine.

In order to perform these tasks, it should happen that:

- User A is authorized to start processes on the server machine;
- The server machine recognizes the authenticity of user A;
- User A recognizes the authenticity of the server machine;
- Process Pa is authorized to read data from the third machine;
- The third machine recognizes the authenticity of process Pa;
- Process Pa recognizes the authenticity of the third machine.

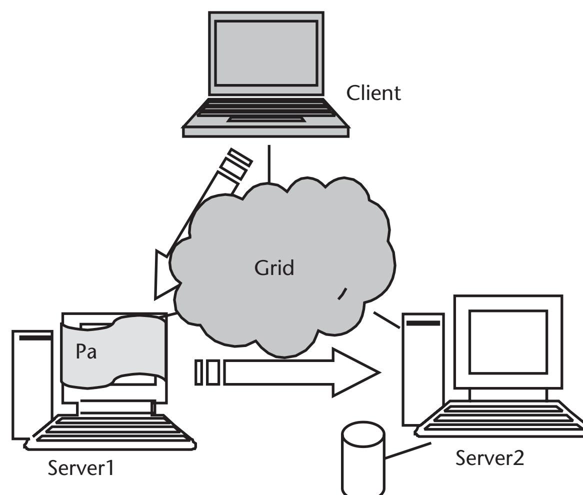

Figure 2.2 In a grid environment, users connect to client machines, from which they start computation on server machines. The launching of an application on a server machine (process Pa in Server1 in the example) can generate the request for the allocation of other resources, eventually located on different machines. In the figure, process Pa, started on the Server1 machine, requests the access to data located on the machine named Server2.

So, each access to a remote resource must be *authorized* and each resource must be *authenticated* in a *mutual* form (i.e., both provider and consumer must authenticate each other when interacting). Authentication and authorization of users are performed via the insertion of a password, as described in detail later. Note also that the resources involved are dynamic (i.e., they change over time in an often unpredictable way) and heterogeneous. When users start up a process, it can in turn request access to other resources, eventually located on different machines of the grid. Each time this happens, a new authentication and authorization session begins. This requires the user to enter the password again. To avoid such a burden, GSI includes the *single sign-on* service, based on the *delegation* of user credentials to the so-called proxies, which act on behalf of the user for a short period of time, as described in Section 2.5.3.

To summarize, GSI guarantees a number of security services, the most important being:

- *Authorization*. This is a mechanism to control the access to resources.
- *Mutual authentication*. Both parties must authenticate (i.e., prove the authenticity of their declared identity) each other in each transaction;
- *Single sign on and delegation*. To eliminate the burden of inserting the password every time access to a resource is needed, the so-called *proxies* act on behalf of users for a limited period of time.

Such security services are now shortly described.

2.5.1 Authorization

The authorization is performed via mapping between global identities and local identities.

Each user is given a *globally unique* name, called *distinguished name* (DN), which identifies it in the context of the grid. A DN contains at least three attributes:

- A user's name or user ID;
- An organization name;
- A country designation.

Other attributes can be entered in order to store additional user and group information. For example, the DNs for two users in the same research group might look like this:

cn=Luciano Tarricone, ou=Electronics, o=Elemgrid, st=Italy, c=I;
cn=Mary Smith, ou=Marketing, o=Elemgrid, st=CA, c=US

In this example, the users work in different departments (ou) for the same group (o). The latter user works in a different state (st) with respect to the former one.

These and other attributes are summarized in Table 2.2.

Table 2.2 Examples of DN's Attributes

Attribute Name	*Syntax*	*Description*	*Examples*
Country	c	Country where the user or group resides	c=US c=GB
Common name or full name	cn	Full name of person or object defined by the entry	cn=Alessandra Esposito cn=printer 3b
Email address	Email	User's or group's email address	Email=alexa@libero.it
Locality	L	Locality where the user or group resides—it can be the name of a city, country, or other geographic region	l=Rome l=Anoka County
Organization	o	Organization to which the user or group belongs	o=University of Naples
Organizational unit	ou	Unit within an organization	ou=Sales ou=Department of Electronics
State or province	st	State or province in which the user or group resides	st=Iowa st=British Columbia

Local identities coincide with local user names as defined on Globus server machines. If the administrator decides to allow access to server resources to a grid user, he must create a userid or choose among the userids available in the server machine. In order to allow the user to access server resources under the chosen userid, the administrator must associate the DN, which identifies unambiguously the user in the grid, to the local userid. Therefore, he must update a mapping table in the server containing the associations between global DNs and local user names (see Figure 2.3).

When a user asks to launch an application on a server machine, the mapping table is checked by Globus daemons. If the user DN appears in the table and is associated to a valid userid, the user is authorized to access server resources under the userid matching his DN.

In Chapter 3, practical guidelines are given to implement the authorization issues.

2.5.2 Mutual Authentication

The GSI adopts the secure sockets layer (SSL) [24] for its mutual authentication protocol. SSL is also known by a new Internet Engineer Task Force (IETF) standard name: transport layer security (TLS).

SSL is based on public-key cryptography [25] and assumes that each entity owns a couple of keys, a private key and a public one (see Appendix B for details on public-key cryptography and all basic security concepts cited in this section). The private key must be kept secret, while the public key must be distributed to other parties. The user's private key is expected by GSI to be stored in a file in the local computer's storage. To prevent other computer users from stealing it, the file containing the key is encrypted via a password (also known as a *pass phrase*). To use the GSI, the user must enter the pass phrase required to decrypt the file containing his private key.

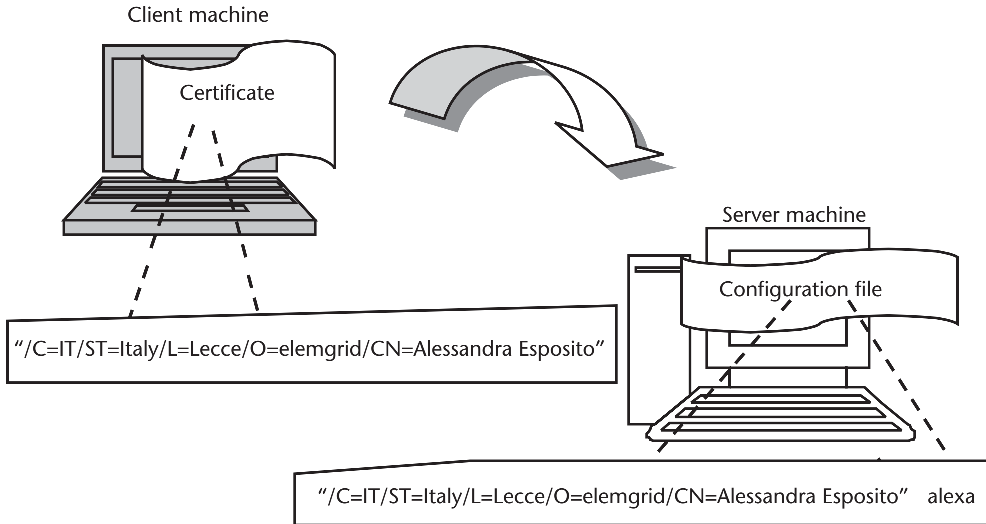

Figure 2.3 In order to allow users access to grid services, client and server machines must be configured to implement authorization polices. On the client side, users should be given an account, a DN ("C=IT/ST=Italy/L=Lecce/O=elemgrid/CN=Alessandra Esposito" in the example), and a certificate associated with that DN. On the server side, a configuration file must be created, where the global DN is mapped onto a local userid ("alexa" in the example). When the end user wants to access resources located in the remote server, he must exhibit his certificate. At the server side, the DN is extracted from the certificate and searched in the authorization file. If the DN is found, the user is authorized to access server resources under the userid matching his DN ("alexa" in the example).

The public key is contained in a *certificate* together with other data:

- Identity of the owner of the certificate (i.e., his DN);
- Expiration date of the key.

To initiate a communication, users exhibit their certificate. A trusted neutral entity, the Certification Authority (CA), must exist to guarantee the authenticity of the association between users and certificates (to defend against the "man in the middle attack" described in Appendix B).

The CA owns a public and a private key as well. The CA uses its private key to sign certificates. When a certificate is created, the administrator must contact the CA and request to sign the certificate. If the CA believes that the certificate is valid, then it signs it.

When a communication session begins, the signature is exhibited together with the certificate, which also contains the identity of the CA (i.e., its DN).

The CA public key is distributed among the parties so that everyone can verify the authenticity of certificates by checking the validity of the CA signature (i.e., decrypting it and comparing the result with the clear text certificate) (see Figure 2.4). Details about digital signing and verification are reported in Appendix B.

So, if two parties have their own certificate, and if both parties trust the CA that signed the counterpart's certificate, then they can prove each other that they are who they are expected to be.

This happens with the SSL protocol steps described here and shown in Figure 2.5:

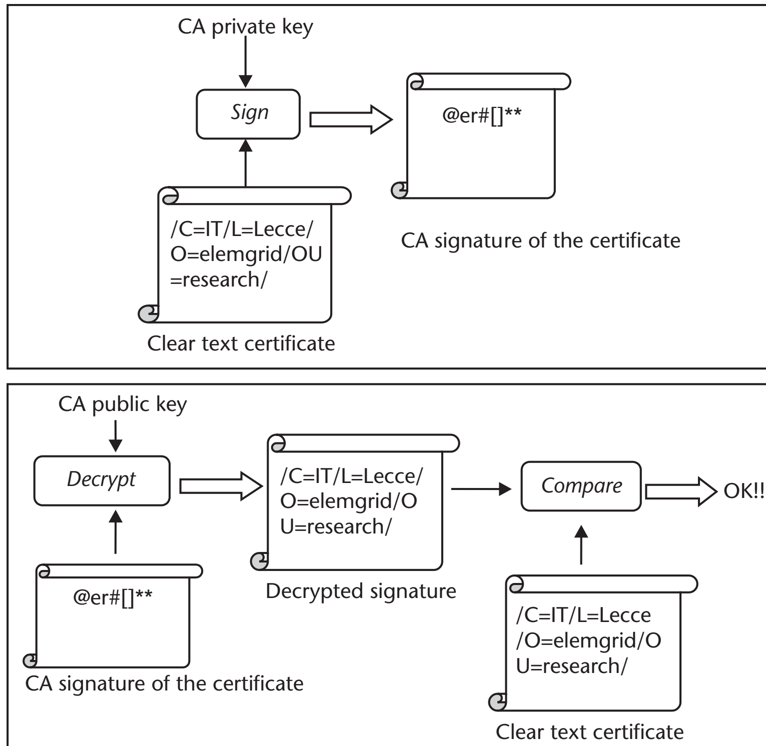

Figure 2.4 Procedure for signing certificates and verifying the validity of the signature. In the first panel, the digital signature procedure is shown: the CA crypts the certificate with its own private key. In the second panel, the signed certificate is first decrypted and then compared with the clear text one. The signature is reputed valid if the decrypted signature matches with the clear text certificate.

1. Party A establishes a connection with the second party (B).
2. To start the authentication process, B gives A its certificate. Party A makes sure that the certificate is valid by checking the CA's digital signature and the expiration date.
3. Party A generates a random message (*nonce*) and sends it to B, asking B to encrypt it.
4. Party B encrypts the *nonce* using his private key and sends it back to A.
5. Party A decrypts the message using B's public key. If this results in the original random message, then A knows that B is who he claims to be.

Now that A trusts B's identity, steps 2 to 5 happen in reverse (i.e., changing A with B and vice versa, so that A can trust B's identity).

At this point, A and B have established a connection each other and are certain that they know their identities: the mutual authentication process has been completed.

2.5.3 Single Sign On and Delegation

The GSI provides a *single sign on* capability: an extension of the standard SSL protocol that reduces the number of times the user must enter his pass phrase. If a grid

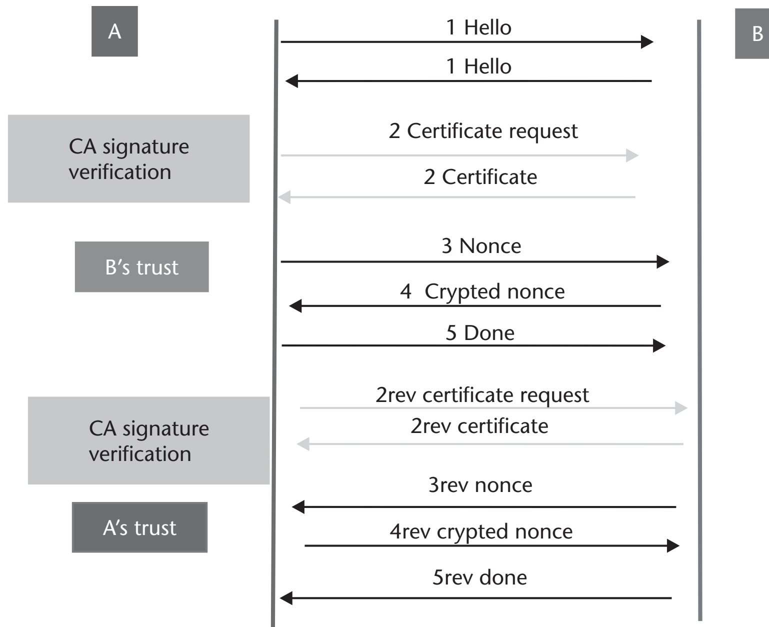

Figure 2.5 GSI uses the SSL protocol for mutual authentication. The protocol consists of a sequence of steps. Steps 2 to 5 authenticate part B to part A. Steps 2rev to 5rev perform the authentication of part A to part B. In step 2 (2rev), part A (B) verifies the validity of the certificate exhibited by part B (A) by checking the CA signature. In steps 3, 4, and 5 (3rev, 4rev, 5rev), part A (B) verifies the authenticity of the identity of B (A) by checking the validity of the encryption performed by part B (A) of a random nonce, sent by part A (B) to part B (A).

computation requires that several grid resources be used (each requiring mutual authentication) or if there is a need to have agents (local or remote) requesting services on behalf of a user, the need to reenter the user's pass phrase can be avoided by creating a *proxy*.

A proxy consists of a new certificate (with a new public key in it) and a new private key. The new certificate contains the owner's identity, slightly modified in order to indicate that it is a proxy. The new certificate is signed by the owner, rather than a CA. For example, if the user certificate contains the following information:

- CA identity: "/O=Grid/O=Globus/OU=elemgrid.org/CN=CA"
- User identity: "/O=Grid/O=Globus/OU=elemgrid.org/CN=Alessandra Esposito"

Then, the proxy certificate contains:

- Identity of the entity certifying the proxy public key: "/O=Grid/O=Globus/OU=elemgrid.org/CN=Alessandra Esposito"
- Proxy identity: "/O=Grid/O=Globus/OU=elemgrid.org/CN=Alessandra Esposito/proxy"

When proxies are used, the mutual authentication process differs slightly. The remote party receives not only the proxy's certificate (signed by the owner), but also

the owner's certificate. During mutual authentication, the owner's public key is used to validate the signature on the proxy certificate. The CA's public key is then used to validate the signature on the owner's certificate. This establishes a chain of trust from the CA up to the proxy through the owner (see Figure 2.6).

As with any private key, the proxy's private key must be kept secure. Nonetheless, proxies have limited lifetimes. Because of this, the security problem related to the proxy's key is less critical than the one being cast by the owner's private key. It is thus possible to store the proxy's private key in a local storage system without encrypting it, as long as the permissions on the file prevent anyone else from easily looking at it. Once a proxy is created and stored, the user can use the proxy certificate and private key for mutual authentication without entering a password.

2.5.4 Other Services

In this section, we have described the fundamental security services provided by GSI: authorization, mutual authentication, and single sign on. Other basic services worth mentioning are *confidentiality* and *integrity*.

By default, the GSI does not establish *confidential* (encrypted) communication between parties: once mutual authentication is performed, the GSI gets out of the way so that communication can occur without the overhead of constant encryption and decryption. Confidentiality is an optional feature, so, if properly configured, the GSI can be used to establish a shared key for encryption.

Integrity is assured when eavesdroppers are prevented from modifying the communication data in any way. Because communication integrity introduces a smaller overhead than encryption, the GSI provides communication integrity by default (it can be turned off if desired).

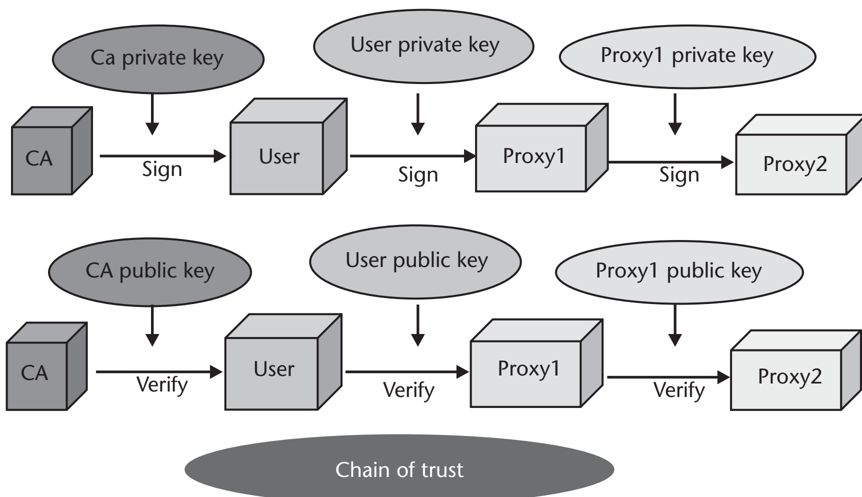

Figure 2.6 Mutual authentication process using proxies. Proxy certificates are signed by their owner (who can in turn be a proxy). When a mutual authentication process occurs, the remote party receives not only the proxy's certificate, but also the owner's certificate. The owner's public key is used to validate the signature on the proxy certificate. The CA's public key is then used to validate the signature on the user's certificate. This establishes a chain of trust from the CA up to the proxy through the owner.

2.6 The Globus Toolkit: The Resource Management Pillar

The RM pillar is responsible for scheduling and allocating resources specifying, for example, resource requirements and the operations to be performed, such as process creation or data access. Core RM services are managed by the *Globus Resource Allocation Manager* (GRAM). Each GRAM is responsible for a set of resources operating under the same allocation policy, often implemented by a local RM system (such as Condor [26], Load Sharing Facility [27], Load Leveler, or Network Queuing Environment). Thus, a computational grid built using Globus typically contains many GRAMs, each responsible for a *local* set of resources. In this manner, individual sites are not constrained in their choice of RM tools. A coallocator distributes requests to GRAMs and manages return values. A schematic overview of RM is proposed in Figure 2.7.

The GRAM service is mainly provided by a combination of two programs: the *gatekeeper* and the *job manager*.

The gatekeeper is the user interface to GRAM. When a job is submitted, the request is sent to the gatekeeper of the remote computer. It authenticates the request using GSI and determines how that user will be authorized locally by mapping it onto a local userid. Then the gatekeeper creates a job manager (see Figure 2.8), which handles the execution of the job as well as any communications with the user. It starts and monitors the remote program, communicating changes of status back to the user on the local machine. When the remote application terminates, correctly or with a failure, the job manager terminates as well.

Applications express resource allocation requests to GRAM via a standard API and a specific language, the *resource specification language* (RSL). The client composes its request in the RSL, which is passed to the job manager by the gatekeeper. The job manager parses the request and translates it into the language of the local scheduler (see Figure 2.9).

RSL allows the users to express the characteristics of the jobs they are going to launch. This can be done via a sequence of *relations*. Relations associate an attribute name with a value (e.g., the relation "executable=a.out" provides the name of an executable in a resource request). By using the relations, users can describe the job to be launched in terms of its environment (e.g., the remote directory where it must work, the standard input, and error and output; see Appendix A for details on these

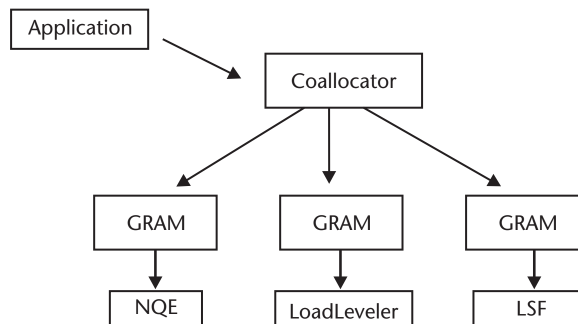

Figure 2.7 Layered architecture of RM pillar. GRAM services are responsible for interacting with local resource managers (NQE, LoadLeveler, LSF in the example), while a coallocator schedules and assigns tasks to the GRAMs.

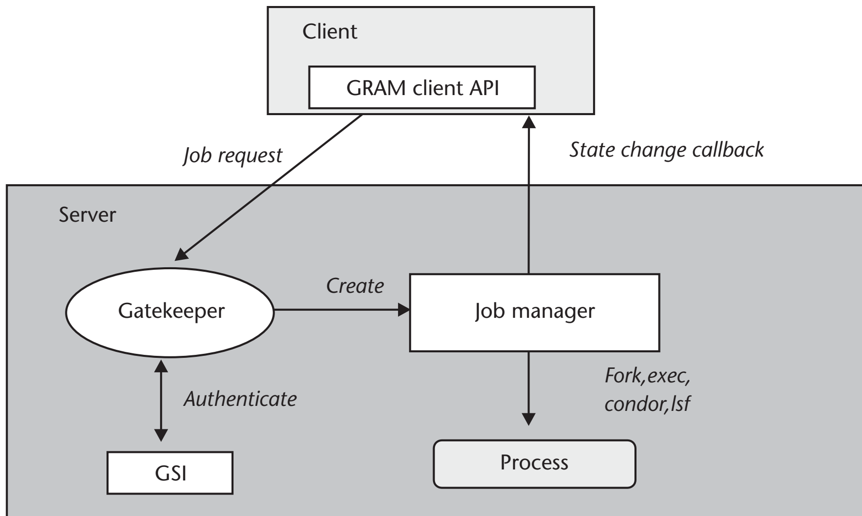

Figure 2.8 The gatekeeper is the user interface to GRAM. When a job is submitted, the request is sent to the gatekeeper of the remote computer. The gatekeeper talks with GSI to authenticate the request. Then it creates a job manager, which handles the execution of the job as well as any communications with the user. The job manager starts and monitors the remote program by calling the local resource manager (i.e., Condor, LSF, or single operating system calls like fork and exec) and then communicates changes of status back to the user on the local machine.

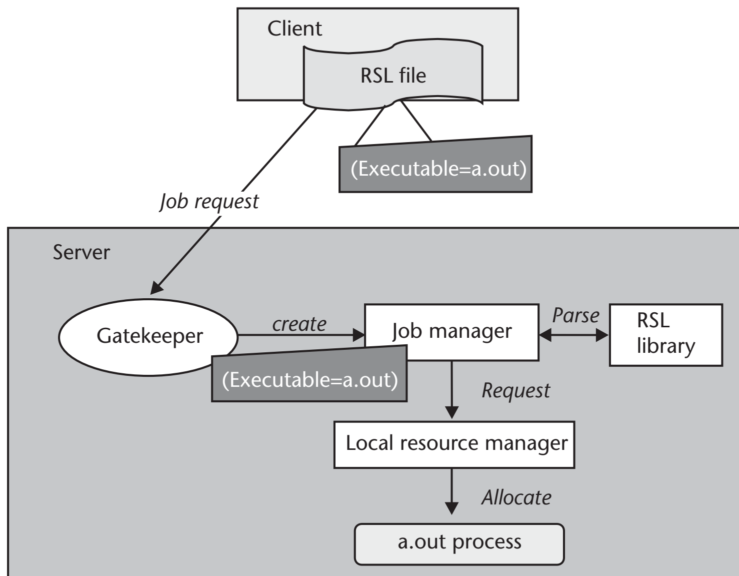

Figure 2.9 Clients can express job requests by writing script files in RSL. The gatekeeper passes RSL instructions to the job manager, which parses them and translates RSL into the language of the local resource manager.

concepts) and of its resource requirements (e.g., maximum amount of memory and maximum CPU time required). In Table 2.3 a list of the most used RSL relations, with their meanings and examples of usage, is reported, while in Chapter 3 examples of real RSL script files are shown.

Table 2.3 Most Common RSL Relations

RSL Relation	*Meaning*	*Example*
(directory=*value*)	Specifies the directory the job manager must use as the default directory for the requested job	(directory=/tmp/bin/)
(executable=*value*)	The name of the executable file to run on the remote machine	(executable=a.out)
(stdin=*value*)	The name of the file to be used as standard input for the executable on the remote machine	(stdin=myfile)
(maxCpuTime=*value*)	The maximum CPU time for a single execution of the executable	(maxCpuTime=60)
(jobType=single|multiple|mpi|condor)	This specifies how the job manager should start the job: "single" starts one process or thread; "multiple" starts multiple processes or threads; "mpi" uses the appropriate method to start a program compiled with a vendor-provided MPI library; "condor" starts jobs in the "condor" universe	(jobtype=single)

Another relevant component of the RM pillar is the *Globus Access to Secondary Storage* (GASS), a service implementing a variety of automatic and programmer-managed data-movement and data-access strategies, enabling programs to read and write remote data. GASS is of fundamental importance in improving the performance of write/read operations onto remote files. Consider, for instance, a sequence of read or write operations from the client machine of Figure 2.10 onto a remote server machine (GASS server).

If the GASS server is not used (scenario 1), the sequence of actions performed when multiple read or write operations are requested by a client applications is:

1. Open the remote file: a file descriptor is returned to the application;

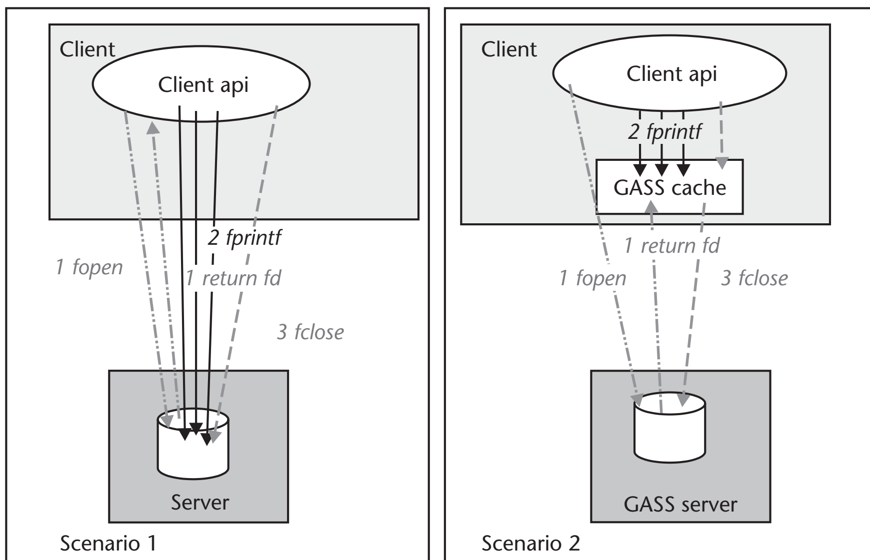

Figure 2.10 Comparing network bandwidth usage with and without the GASS server.

2. Write and/or read data onto the remote file;
3. Close the remote file.

Step two can be burdensome, as data must travel in the network.

Scenario 2 shows how GASS uses a file cache, a local storage area where copies of remote files are stored. By default, data are moved into and out of this cache when files are opened and closed. In this manner, programs perform their write and read operations on the local copy of the remote file, thus minimizing the problems related to the limited availability of network bandwidth. In scenario 2, the sequence of actions becomes:

1. Open the remote file: the file is copied in the local cache and a file descriptor is returned to the application;
2. Write and/or read data onto the local copy of the remote file;
3. Close the file: the local file is copied back to the remote location.

Step two is now totally performed on the client machine, without using network bandwidth.

Instead of accessing the remote file every time a write/read operation is performed, GASS uses a file cache. By default, data are moved into and out of this cache when files are opened and closed.

When a remote file is opened by using the appropriate GASS API function, the following actions are taken:

1. The file is looked up in the local cache;
2. If it does not exist, the file is copied from the remote server into the local cache;
3. The local file is opened.

When the file is closed, the file is copied from the local cache onto the remote server.

In this manner, programs perform their write and read operations on the local copy of the remote file, thus minimizing the problems related to the limited availability of network bandwidth.

The GASS service is integrated with other GT components: GRAM uses GASS mechanisms to allow both executables and standard input, output, and error to be identified by using uniform resource locators (URLs) in RSL files. For example, if a RSL script contains a relation like the following:

(executable=http://www.mickey.org/a.out)

then, before executing the job, the file named "*a.out*" is transferred from the machine named "*www.mickey.org*" to the GASS cache of the remote machine where the job must run. The copy of the executable file "a.out" is removed after the job has terminated. This feature allows a user to ask that an application resident on a remote machine can run on a different remote machine. The process is sketched in Figure 2.11.

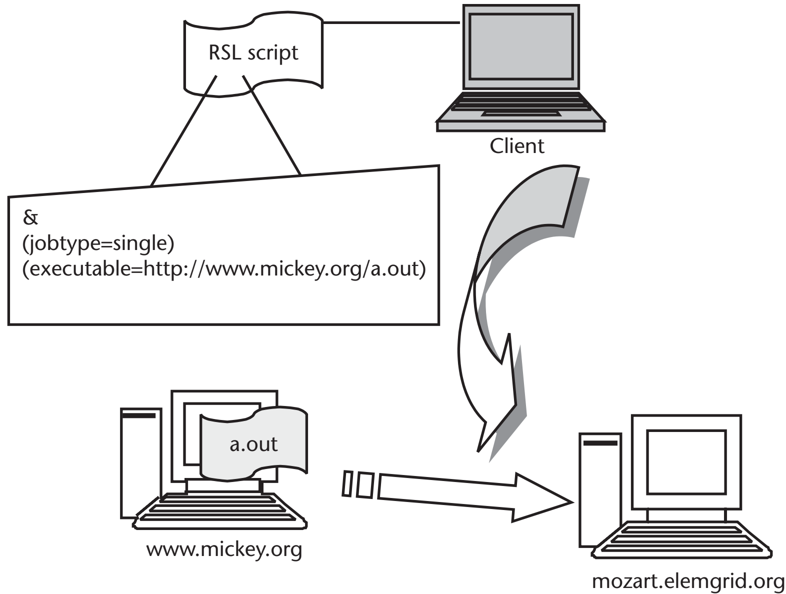

Figure 2.11 The RSL syntax allows users to ask that the application resident on a remote machine is executed on a third machine. In the example, the end user logs into the client machine, where he writes an RSL script file asking that the application named "a.out", resident on "www.mickey.org", is launched on the server machine named "mozart.elemgrid.org".

2.7 The Globus Toolkit: The Information Services Pillar

The IS pillar provides information about the structure and state of resources (e.g., their current load and usage policy). It can answer questions about the system state such as:

- What resources are available?
- What is the state of the computational grid?
- How can we optimize an application given the available resources and their current state?

The *metacomputing directory service* (MDS) is the IS core. MDS has a distributed architecture. It is basically composed of *grid resource information services* (GRIS) and *grid index information systems* (GIIS). Each MDS resource can run its GRIS, which is able to respond to queries from other systems of the grid about the status of the resource (e.g., amount of disk space, amount of memory, or number and speed of processors). A GRIS can be configured to register itself to aggregate directory services (such as a GIIS). GIIS gather data from the registered GRIS to provide aggregate reports about the status of portions of the grid (see Figure 2.12).

GRIS and GIIS share a common *directory service* (i.e., a common protocol to store and retrieve resources in a distributed environment) and use the same *information model* (i.e., they adopt the same structure to represent resource data). In the following subsections, the directory service and the information model are shortly described.

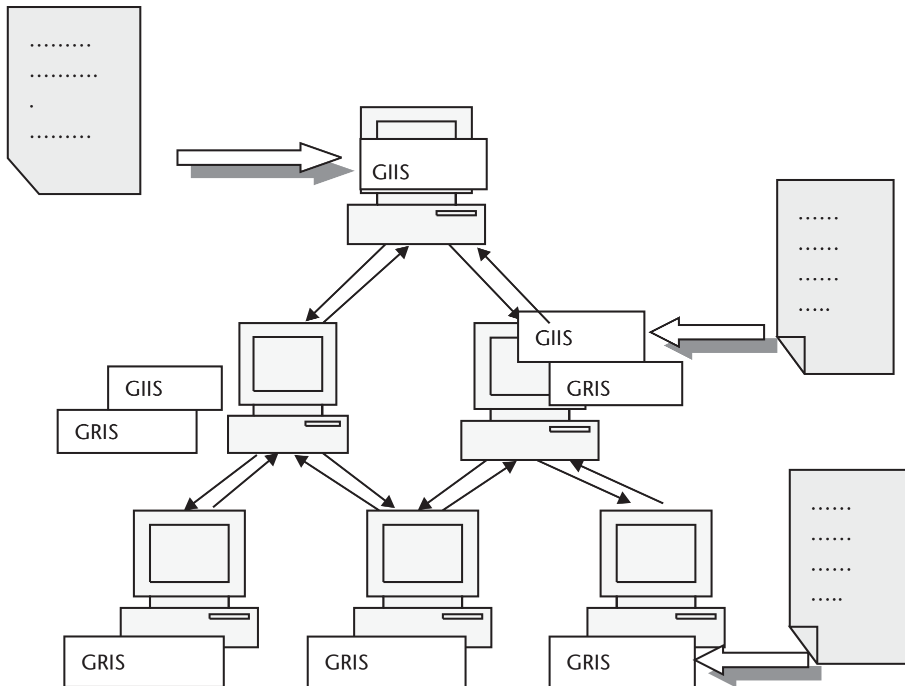

Figure 2.12 MDS distributed architecture. Each node hosting a GRIS provides data on local resources, while machines hosting GIIS play the role of data collectors. Queries about the state of resources can be addressed to single GRIS to obtain data related to a single host or to GIIS to obtain aggregate data related to a group of machines.

2.7.1 MDS Directory Service: Lightweight Directory Access Protocol

The MDS directory service is the *lightweight directory access protocol* (LDAP) [28, 29]. MDS components contact LDAP servers to create, retrieve, and modify data about grid resources.

LDAP was designed to store small records of information in a hierarchical tree structure. This structure resembles the tree of a file system, with nodes containing attributes and connecting with subtrees. Starting at a root node, the LDAP tree of information, known as the directory information tree (DIT), contains a hierarchical view of all of its data and provides a tree-based search system for the data.

Objects are named by their position in the tree, just like directories are called in a file-system tree. For computational grids, the root of the tree is usually named "o=grid," where "o" stays for organization. The DIT branches down this root, adding organizations, domain components ("dc"), organization units ("ou"), resources, and so on (see Figure 2.13). Every node in the DIT structure has a unique path to the root. That path serves as an unambiguous name to the entry associated with that node. This unambiguous name is the DN of the entry, which is used to locate the resource data.

2.7.2 MDS Information Model

The MDS information model represents data through a hierarchical structure. In this structure, each entity is given a name and a list of attributes. When a query is

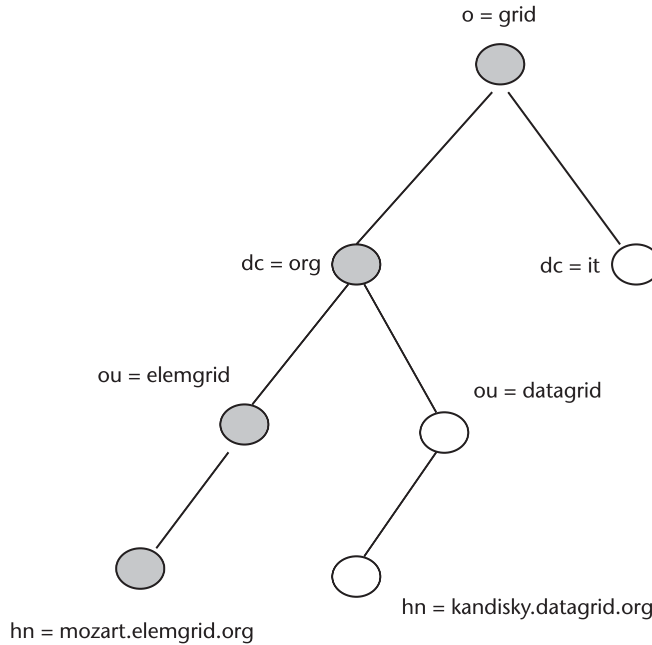

Figure 2.13 Tree structure of LDAP information. Starting from root (o=grid), the tree branches down to leaf nodes. The unique path from the root to the leaf defines the globally unique name of resources. Given the global name of a resource, a tree search algorithm is used to locate its data in the grid environment. LDAP contains the tools to add branches to the tree, extending a grid to organizations, organization units, and so on. In the example, two domain components (as defined in the Internet naming conventions) are shown. The "org" domain component contains two organization units, each pointing to a host resource. For example, the host named "mozart.elemgrid.org" is identified by the DN "o=grid/dc=org/ou=elemgrid/hn=mozart.elemgrid.org."

performed, the object is located through the LDAP protocol and the attribute values are returned.

Hosts are represented as a collection of hardware (e.g., CPU or memory) and software (e.g., operating system) devices organized in a tree structure (see Figure 2.14).

This structure groups similar devices (i.e., multiple processors or different memory devices, such as RAM and virtual memory) under the same node, which branches down to the single devices. This allows making aggregate queries on groups of devices.

The root of the structure is represented by the host, identified by its hostname, while single devices are leaf nodes. Each device is associated with a list of attributes (e.g., CPU model or RAM size and free space), whose values are returned when a query arrives at MDS.

For example, the attribute identifying the name of a mode is:

- The *hostname*, if the node is a root node;
- The value of the attribute called *device-group-name,* if the node represents the aggregation of devices;
- The value of the attribute called *device-name,* if the node represents a single device.

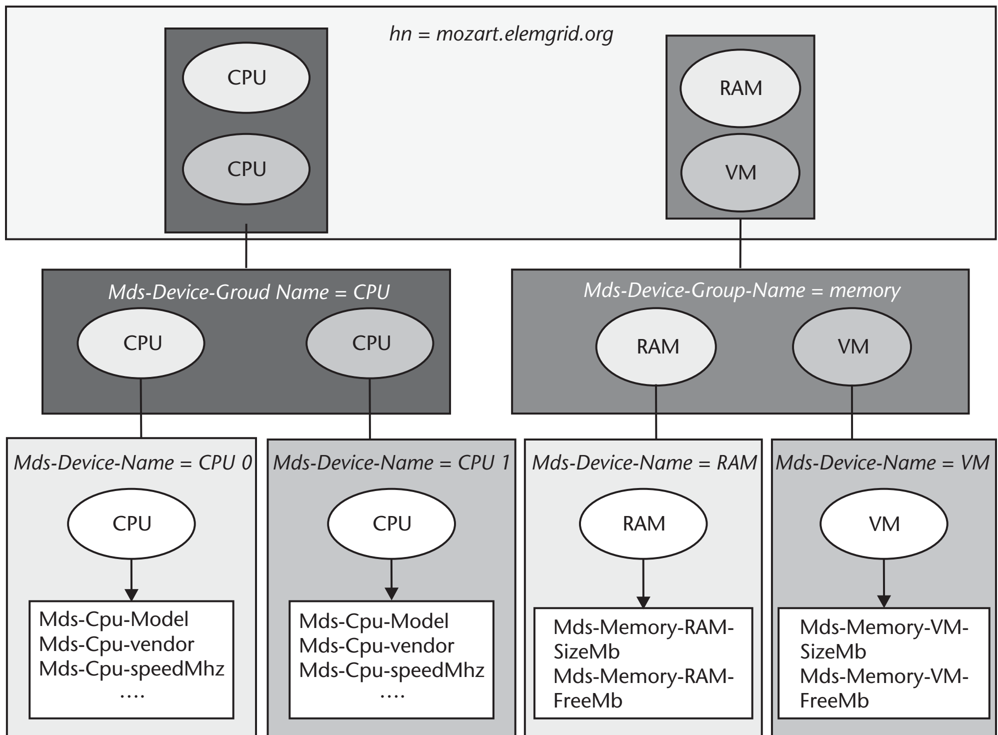

Figure 2.14 GRIS information model: GRIS report data about hosts they are installed on. Data are represented by a well-defined information model, which includes the most relevant devices a host may contain. The information model represents host data through a hierarchical structure that groups similar devices such as multiple processors or different memory resources such as random-access memory (RAM) and virtual memory under the same *device-group-name.* At the root of the structure, there is the host, identified by its hostname ("mozart.elemgrid.org" in the example), while single devices are at the leaf nodes. Each device is associated with a list of attributes (e.g., CPU model or RAM size and free space), whose values are returned when a query arrives at the GRIS. This tree structure facilitates the use of filters. Queries may contain filters to select the attributes to be returned. If filters are not used, a standard query returns all data related to the host; otherwise, the user can request information on a single device (e.g., RAM), a specific attribute of a device (e.g., the model of the CPU), or groups of devices (e.g., data on processors belonging to the host).

Other relevant attributes are:

- Mds-Cpu-speedMhz;
- Mds-Cpu-Model;
- Mds-Cpu-vendor.

These contain speed, model, and vendor of processors. Also:

- Mds-Memory-RAM-SizeMb;
- Mds-Memory-RAM-FreeMb.

These refer to RAM total and available space.

The tree structure of the MDS information model renders the use of filters simpler. Queries may contain filters to select the attributes to be retrieved. If filters

are not used, a standard query returns all data related to the host, otherwise the user can request information on a single device (e.g., RAM memory), a specific attribute of a device (e.g., the model of the CPU) or groups of devices (e.g., data on all the processors belonging to the host).

The query composition can be greatly simplified by using an LDAP browser. A number of LDAP browsers exist, most of them freely downloadable from the Internet [30], making the navigation through MDS information structure easier.

2.8 The Globus Toolkit: The Data Management Pillar

The DM pillar performs two fundamental tasks.

The first (*distributed data access and management*) is mostly related to data movement: in order to run a job, the end user should be able to access remote data, transfer input data to the target machine, and copy back the resulting data sets. The second (*dataset replicas services*) is related to the need to replicate huge datasets among a number of connected storage systems. In the following subsections, these tasks are briefly overviewed.

2.8.1 Distributed Data Access and Management

Distributed scientific and engineering applications often require access to huge amounts of data (up to the terabyte or petabyte range). When large amounts of remote data must be accessed, speed, security, and reliability become key factors. A number of distributed storage systems have been developed to cope with these and other related issues. Among them, it is worth mentioning HPSS [31] from IBM, DPSS [32] from LBNL, and SRB [33] from San Diego Supercomputer Center (SDSC). Each of them meets specific requirements related to the access and management of distributed huge datasets. For instance, HPSS and DPSS focus on data transfer performance by transferring parallel data streams, while SRB focuses on hiding heterogeneity among different storage resources (such as file systems, DBMS objects, and tape archives) by providing transparent access via a uniform API. These systems are proprietary and mostly tied to specific client software. Because of this, the Globus team developed a middleware software to promote interoperability among existing and future storage systems. This software was designed with the same philosophy as other GT components:

1. A bag of basic services satisfying all of the basic requirements emerged in distributed communities;
2. A general API to promote interaction.

To achieve this ambitious goal, the Globus team implemented an extended version of the ftp, GridFTP, which adds a series of features to ftp, customizing it to grid environments. The main features are:

- *Partial file access*. This feature is very useful when dealing with huge files, because in these cases bandwidth can be saved, provided that only the needed portions of the files are moved.

- *Secure transfer*. With GSI and Kerberos [34] support, GridFPT provides authentication, privacy, and integrity check services;
- *Parallel transfers*. The parallel movement of transfer control protocol (TCP) streams facilitates high-speed transfers and permits a considerable bandwidth saving.
- *Third-party transfers*. GridFTP includes an authenticated protocol to permit third-party control of transfers between two remote dataset storage systems.
- *Reliable file transfer*. GridFTP furnishes fault recovery methods to cope with transient network failure and server outages and to restart failed transfers.

2.8.2 Dataset Replicas Services

When optimization of data access times is the most critical issue, it can be useful to create a number of dataset replicas (i.e., to generate identical copies of data and store them in different sites). This can reduce data access latency. The creation of data replicas can considerably improve the performance of data access, but it adds a number of complications not existing when dealing with a single instance of files. For example, replicas location must be tracked and associated with each other (*replica management*), and users should be enabled to access replicas transparently, eventually by specifying a selection criterion (*replica selection*).

Replica management tools allow the creation or deletion of replicas on storage sites. They typically maintain a *replica catalog* containing information about stored datasets and replica site addresses. Each dataset has a unique logical name that corresponds to a number of physical replicas, geographically distributed in the grid. The catalog stores correspondences between logical names and physical locations. GT offers both an API to manipulate data in replica catalogs and an API to perform basic replica management tasks.

Replica selection is the process of choosing a replica among those spread across the grid, based on some characteristics specified by the application. One common selection criteria is access speed. This way, the introduction of distributed data redundancy is adequately exploited. GT includes the basic services to perform replica selection by associating a GRIS to each storage resource. In this way, storage resources publish their attributes (such as storage capacity and seek time), which can be queried and used by storage brokers (i.e., the tools dedicated to the selection of the replica), given the application requirements.

Given the previously mentioned API, it is possible to develop an application performing the following tasks:

- Query a local archive to choose the logical name of the dataset to access;
- Contact a storage broker and communicate the requirement;
- The storage broker matches application requirements against available storage resource attributes and returns the address of the best dataset;
- The application accesses the stored data.

2.8.3 Conclusions

It is worth mentioning that Globus efforts in the fields of data management are currently appreciated in a number of outstanding projects.

One is the Teragrid project [35], launched by the National Science Foundation [36] in August 2001 and joined by five groups, among which we remember the National Center for Supercomputing Applications (NCSA) [37] and Argonne National Laboratory [38]. Among other activities, Teragrid worked for the extension of SDCD's SRB to GridFTP. The implementation of specific drivers to GridFTP allowed SRB to expand the number of heterogeneous systems that users can connect to by a SRB client.

It is also worth mentioning the European DataGrid project [39]. DataGrid was funded by CERN [40] and includes leading European scientific partners such as ESA/European Space Research Institute (ESRIN) [41] (Italy) and Centre National de la Recherche Scientifique (CNRS) [42] (France). DataGrid joins groups involved in data-intensive applications that are interested in sharing huge amounts of distributed data over the network infrastructure. The project experiments with current middleware technologies and contributes to their improvement by working in strict cooperation with the Globus Team.

2.9 The Globus Toolkit API

All of the Globus components offer an API accessible by both programs written in C language and in Java language, thus allowing the use of Globus utilities in the context of an application. APIs facilitate the development of programs by allowing the use of functionalities embedded in software or hardware tools inside the same programs. A tool contains an API when it defines a number of function calls (*interfaces*) to access its own facilities. Interfaces are characterized by rigorous and permanent specifications and standards. The existence of interfaces, hiding all of the implementation details to the software designer, allows the development of programs without the need for intimate knowledge of the device or software with which the application interacts.

Globus API provides libraries for embedding the three pillars and GSI facilities inside applications, plus a number of utilities for:

- *Communication.* Communication facilities are supported by a dedicated library, called Nexus. The Nexus library provides the communication facilities required to implement compilers for advanced languages, libraries, and applications in heterogeneous parallel and distributed computing environments. Systems that rely on Nexus mechanisms include compilers for the parallel languages CC++ and HPC++, as well as the MPICH implementation of the MPI standard (see Section 2.10 for further details on MPICH).
- *Coallocation.* Globus contains facilities useful for distributing jobs among the available resources, when several GRAMs are installed (see Figure 2.7). These facilities are embedded in the Dynamically Updated Request Online Coallocator (DUROC). It offers an API to execute jobs to be distributed over resources accessed through independent GRAMs. DUROC parses RSL requests to

determine how a job might be distributed across the resources of which it is aware, then chooses the resources to allocate, and finally issues job requests to each of the pertinent GRAMs to schedule the job.

- *Fault detection*. The Heartbeat Monitor (HBM) library allows users to monitor remote processes and to intercept notification of exceptions. Through a client interface, a process is allowed to register itself with the HBM service and to send regular heartbeats to it. Moreover, a data collector API allows a process to obtain information related to the status of other processes registered with the HBM service, thus supporting, for example, fault recovery mechanisms.

Globus also provides *Commodity grid* (*CoG*) *kits,* which allow users to use *commodity* (i.e., enabling) frameworks, technologies, and toolkits in cooperation with grid technologies. CoG kits include, for instance, Java, Java servlets, CORBA, and Matlab.

2.10 The MPI with Globus

As discussed in Chapter 1, message passing is a leading paradigm in distributed and parallel computing, with MPI [43, 44] being the international standard specification. It is progressively replacing the other parallel protocols, though in some cases (for instance, when spawning tasks dynamically is required), it is not suitable and alternative solutions, such as PVM [45], must be adopted.

The Message Passing Interface Forum (MPIF), with participation from over 40 organizations, has been meeting since November 1992 to discuss and define a set of library interface standards for message passing. The result is MPI, a set of general guidelines to support message passing on whatever platform is suited to parallel and distributed computing.

MPI is well suited for SPMD parallel paradigms. According to this paradigm, all of the processors execute the same program. Each processor may run more than one process. The number of processes to be run is given by the user when launching the executable. The executable must be installed and compiled on each platform before executing. It is also worth mentioning that MPI is quite flexible and works well for multiple program multiple data (MPMD) programming too.

MPI defines a number of procedures to implement parallel programs. In MPI, the most attractive features of a number of existing message passing systems are used:

- *Point-to-point communication*. MPI defines the basic operations to allow processes to send and receive messages.
- *Process groups*. MPI contains the facilities to define groups of processes (i.e., processes sharing a common tag so that they can be addressed all together by a set of operations, called *collective* operations, which are briefly introduced later).
- *Collective operations*. MPI includes the functions to make processes belonging to the same group communicate simultaneously. The most relevant

functions are *barrier synchronization*, which is used to synchronize group members; *broadcast* to send a message to all of the members of a group; and *gathering,* which is used to gather data spread among members.

- *Communication contexts.* MPI provides the ability to have separate safe "universes" of message passing, so that communication internal to a library execution is prevented from external communication.
- *Process topologies*. The definition of *virtual* process topologies allows users to assign names to the processes of a group that reflect the logical communication pattern of the process (usually determined by the underlying problem geometry and the numerical algorithm used). The virtual process topology can be exploited by the system in the assignment of processes to physical processors, if this helps in improving the communication performance.

The suitability of MPI for developing parallel algorithms in EM has been demonstrated in a recent paper [46].

If MPI is the reference standard for message passing, one dedicated tool for MPI inside heterogeneous distributed systems is represented by MPICH [47].

MPICH is a *portable* implementation of the MPI specification. It is designed to be ported and optimized for a variety of systems through implementations of an abstract device interface (ADI). Each implementation of an ADI is called *device* (example of devices are those developed for Beowulf clusters or shared memory systems).

A derivation from MPICH suited to grid environments based on GT is MPICH-G2 [47], which is our reference library for MPI in grid environments. MPICH-G2 is the MPICH device developed to provide a grid-enabled implementation of MPI. That is, using Globus services (e.g., job startup, security), MPICH-G2 allows users to couple multiple machines belonging to a grid and run MPI applications.

As suggested in [47], "One important class of problems where MPICH-G2 fits well is composed of those that are *distributed by nature* (i.e., problems whose solutions are inherently distributed). One example is remote visualization applications, in which computationally intensive work producing visualization output is performed at one location and the images are displayed on a remote high-end device.

A second class of problems consists of those that are *distributed by design*, where there is access to multiple computers, perhaps at multiple sites connected across a WAN, and a user may want to couple these computers in a computational grid.

In another scenario, a user may have an MPI application that runs on a single MPP but has problems that are too large in size for any single machine to which the user has access. In this situation, a grid-enabled implementation of MPI like MPICH-G2 may help by enabling the user to couple multiple MPPs in a single execution." After installing MPICH-G2 on grid nodes, the application can be ported straightforwardly to run in the grid environment: the migration towards MPICH-G2 of an MPI application does not require any changes to the code, as described in detail in Chapter 3.

2.11 Dedicated Tools: Economy-Driven RM in Grids

Grids federate a multiplicity of resources distributed among diverse organizations. Most existing grids gather resources belonging to *scientific* organizations, which pool them into the grid without expecting any financial earning. By nature, they obtain benefits from the scientific results of grid computing. The exploitation of grid computing can be further enhanced if *commercial* organizations are encouraged to get involved in grid communities. Commercial organizations would be surely interested in profiting from the rent of resources to grid communities.

The most widespread grid middleware technologies, such as Legion and Globus, do not support any economy-based computing model. Thus, they must be integrated with tools supporting computational economy. These tools are the *economy-driven resource managers* (i.e., they are resource managers with the capability to select target resources based on price, objective, and constraints of users, with time or budget being perhaps the most typical).

The most outstanding infrastructure for economy-driven grids is GRACE [48], which finds in Nimrod-G [49] the reference tool for trading resources. The GRACE infrastructure is a middleware component that coexists with middleware systems (such as Globus) to support computational economy. GRACE employs a *competitive market*, where the client tries to minimize the cost of computation for resource users and the server tries to maximize the profit for resource owners. GRACE includes a protocol to allow client and servers to negotiate the cost of resources until one of them says that its offer is the last one. If the other party accepts the deal, then both work together, on the basis of the agreement reached in the negotiation phase.

Nimrod-G [50] is an RM tool coordinating the access to grid resources in cooperation with local schedulers (e.g., Condor and LSF) via grid middleware services (e.g., Globus). Nimrod-G can play the role of the coallocator drawn in Figure 2.8. When the application is submitted to the tool for execution, the user can specify the deadline that results are needed by and the maximum cost he can support. Grid resources must be listed and communicated to the tool, specifying their attributes, including the cost (which can be dynamic, if the GRACE infrastructure is implemented). Based on such information, Nimrod-G allocates the resources with the goal of optimizing the cost or the application performance, as selected by the user. However, the grid resource availability and load vary over time, so Nimrod-G continuously monitors the state of resources, changing those dedicated to the submitted experiment if it understands that the deadline cannot be met with the current resource set.

2.12 Web-Based Technologies and Projects

As seen in Chapter 1, grids can be used for a variety of purposes, starting from cooperation in software development to exploitation of idle CPU cycles for HPC. A number of projects, each focusing on a different role and context of application of GC, have been developed and are still in progress. Most of them rely on middleware software for the basic services, with GT being the most used, and use the Web interface to communicate.

WebFlow [51, 52] is a Web-based visual tool for development of grid-enabled applications. It integrates OO technologies and Web tools to create high-level programming environments and to support distance computing on heterogeneous distributed platforms. Webflow users connect to the grid via the uniform Web client interface (i.e., the Web browser) and develop distributed applications by assembling reusable components or by developing new components. They define the dataflow between the application components by drawing tagged links among the components. The components may be located and launched anywhere in the network. Webflow is a perfect example of integration between different enabling technologies and middleware services: it talks to CORBA when interfacing with Intranet components, uses Globus middleware services to locate resources and launch jobs in the grid, and is based on Java technology as it uses Java servlets to manage and coordinate the various system components.

The Cactus Code [52, 53] provides a collaborative environment for the development and running of scientific software. Cactus allows engineers, scientists, and software developers to integrate their competences and plug their work into the same application, even when located far from one another. Cactus is focused on scientific software, being interfaced with the most used scientific numerical libraries. In a number of scientific applications, the resources required are not known a priori (think of the technique called *adaptive mesh refinement*, where the resolution of a computational mesh can be increased while running). In these cases, Cactus can be configured to automatically seek out additional resources while running the application, eventually migrating the code or launching new subtasks.

Web portals are another very popular technology used for GC. The first examples of portals are represented by well-known applications, such as Yahoo, Excite, and Google. They represent a useful, easy "entry point" for Web users to a number of fundamental services, such as searching in the Web, e-commerce, or Web marketing. As explained in Chapter 1, the Web is more and more evolving towards interaction and programmatic computing. An example of this ongoing trend is given by Google, one of the most used search engines. Google offers an API to access its index programmatically. In this manner, Google archives can be queried by software applications that can continuously access Web data to perform market researches or get updated information on preferred subjects.

In the case of distributed systems, portals are intended to package the system components of a grid application under a single Web server, mediating the scheduling and control of grid resources. The penetration of Web portals leads to the definition of the *Object Web* [54], where Web technology integrates with OO paradigms and GM tools to form grid-enabled applications exploitable by the uniform Web interface. NPACI Hot-Page [55] GC portal and the Astrophysics Simulation Collaboratory [56] are the most well-known examples of this trend. The Astrophysics Simulation Collaboratory, for instance, integrates Java technology, Globus middleware, and Cactus Code framework to allow scientists develop and run simulations in grid environments via the Web browser.

2.13 Grid-Enabled HTC: Condor-G

HTC environments are computing environments that deliver large amounts of computational power over a long period of time. HTC environments try to optimize the number of jobs they can complete over a long period of time.

A widespread tool to maximize the throughput in distributed environments is Condor [26]. Condor works to exploit idle CPU cycles of connected machines. It provides transparent checkpointing and restart facilities so that computations can be migrated from overloaded or failed machines to lightly loaded ones. Condor's functionality, called DAGMan, manages the submission of a large number of jobs with simple or complex dependencies on one another. Users may specify preprocessing and post processing of jobs and represent dependencies by a direct acyclic graph (DAG). In a DAG, the programs are represented by nodes in the graph, and the edges identify the dependencies. For example, a DAGman file may contain the following lines:

```
PARENT A CHILD B C
PARENT B C CHILD D
```

They specify that:

- Jobs B and C depend on A (i.e., they must start when A has completed);
- Job D depends on jobs B and C (i.e., D can start only when both B and C have been completed).

Current Condor limitations (such as its ability to migrate processes only within its server pool) make it work well in single administrative domain systems. Condor-G [57] extends Condor to grid environments by merging features of Condor (i.e., its ability to harness distributed computational power to maximize throughput) with GT characteristics (i.e., the openness and flexibility of its protocols and services, particularly in the fields of security and resource management). By submitting a DAG file to Condor-G, users can harness the computational power of grid resources as if they belonged to a single domain network. Users can submit many jobs at once and then monitor running jobs with a convenient interface, receive notification when jobs complete or fail, and maintain Globus credentials if they expire while a job is running.

References

[1] Baker, M., R. Buyya, and D. Laforenza, "The Grid: International Efforts in Global Computing," *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, Italy, 2000.

[2] Khoshafian, S., and R. Abnous, *Object-Orientation: Concepts, Languages, Databases, User Interfaces*, New York: John Wiley, 1995.

[3] Booch, G., *Object-Oriented Analysis and Design (With Applications),* Redwood, CA: Benjamin-Cummings Publishing Co., Inc., 1994.

[4] Nicol, J. R., C. Thomas Wilkes, and F. A.Manola, "Object Orientation in Heterogeneous Distributed Systems," *IEEE Computer*, June 1993, pp. 57–67.

[5] DCE, http://www.opengroup.org/dce.

[6] Thai, T. L., A. Oram, *Learning Dcom*, Sebastopol, CA: O'Reilly & Associates, April 1999.

[7] http://www.omg.org.

[8] Monson-Haefel, R., *Enterprise JavaBeans*, Sebastopol, CA: O'Reilly & Associates, October 2001.

[9] Oaks, S., and H. Wong, *Jini in a Nutshell*, Sebastopol, CA: O'Reilly & Associates, 2000.

[10] Siniaris, C. G., et al., "Implementing Distributed FDTD Codes with Java Mobile Agents," *IEEE Antennas and Propagation Magazine* , Vol. 44, No. 6, December 2002, pp. 115–119.

[11] Kafura, D., *Object-Oriented Software Design and Construction with Java*, Englewood Cliffs, NJ: Prentice-Hall, 2000.

[12] Liotta, G., M. Mongiardo, and L. Tarricone, "Introductory Review on Object Oriented Paradigm for Full-Wave Microwave CAD," *International Journal on Radiofrequency and MW CAE*, Vol. 12, 2002, pp. 341–353.

[13] Felsen, L. B., M. Mongiardo, and P. Russer, "Electromagnetic Field Representations and Computations in Complex Structures I: Complexity Architecture and Generalized Network Formulation," *International Journal on Numerical Modelling*, Vol. 15, 2002, pp. 93–107.

[14] Felsen, L. B., M. Mongiardo, and P. Russer, "Electromagnetic Field Representations and Computations in Complex Structures II: Alternative Green's Functions," *International Journal on Numerical Modelling*, Vol. 15, 2002, pp. 109–125.

[15] Felsen, L. B., M. Mongiardo, and P. Russer, "Electromagnetic Field Representations and Computations in Complex Structures III: Network Representations of the Connection and Subdomain Circuits," *International Journal on Numerical Modelling*, Vol. 15, 2002, pp. 127–145.

[16] Cristoffersen, C. E., U. A. Mughal, and M. B. Steer, "Object-Oriented Microwave Ciruict Simulation," *International Journal on Radiofrequency and MW CAE*, Vol. 10, 2000, pp. 164–182.

[17] Olyslager, F., et al., "An Academic FDTD Simulator Using Object Orientation," *AP2000 Int. Conference,* 2A1.2, Davos, Switzerland, April 9–14, 2000.

[18] IDS Technical Report Design Framework Builder V2.0, System Integrator's Manual, RT/97/030, Ingegneria dei Sistemi spa, Pisa, Italy.

[19] Titomanlio, S., "Antenna Design Framework for Large Array Design," *SAR CAD Rev. Meeting 1*, ESTEC, NL, September 1998.

[20] Legion, http://legion.virginia.edu.

[21] Globus, http://www.globus.org.

[22] Foster, I., and C. Kesselman (Eds.), *The Grid: Blueprint for a New Computer Infrastructure,* San Francisco, CA: Morgan Kaufmann, 1999.

[23] Foster, I., C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int. Journal of High Performance Computing Applications*, Vol. 15, No. 3, 2001, pp. 200–222.

[24] http://www.openssl.org.

[25] Menezes, A., P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, Boca Raton, FL: CRC Press, 1996.

[26] Condor, http://www.cs.wisc.edu/condor.

[27] LSF, http://ww.platform.com.

[28] Howes, T., and M. Smith, *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol,* New York: Macmillan Technical Publishing, 1997.

[29] http://www.openldap.org.

[30] LDAP, http://www-unix.mcs.anl.gov/~gawor/ldap.

[31] HPSS, http://www.sdsc.edu/hpss.

[32] DPSS, http://www-didc-lbl.gov/DPSS.

[33] SRB, www.sdsc.edu/DICE.

[34] Steiner, J., B. C. Neuman, and J. Schiller, "Kerberos: An Authentication System for Open Network Systems," *Proc. Usenix Conference*, Dallas, TX: 1988, pp. 191–202.

[35] TeraGrid, http://www.teragrid.org.

[36] NSF, http://www.nsf.gov.

[37] NCSA, http://www.ncsa.uiuc.edu.

[38] http://www.anl.gov.

[39] DataGrid, http://www.eu-datagrid.org.

[40] CERN, http://www.cern.ch.

[41] ESA/ESRIN, http://www.esa.int.

[42] CNRS, http://www.cnrs.fr.

[43] Pacheco, P. S., *Parallel Programming with MPI*, San Francisco, CA: Morgan Kaufman, 1997.

[44] http://www.mcs.anl.gov/mpi.

[45] Dongarra, J, et al., "Integrated PVM Framework Supports Heterogeneous Network Computing," *Computers in Physics*, April 1993.

[46] Guiffaut, C., and K. Mahdjoubi, "A Parallel FDTD Algorithm Using the MPI Library," *IEEE Antennas and Propagation Magazine*, Vol. 43, No. 2, April 2001, pp. 94–103.

[47] http://www.mcs.anl.gov/mpi/mpich/download.html.

[48] Buyya, R., D. Abramson, and J. Giddy, "An Economic Driven Resource Management Architecture for Global Computational Power Grids," *Intl. Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '2000)*, Las Vegas, NV, June 2000, pp. 26–29.

[49] Buyya, R., D. Abramson, and J. Giddy "Nimrod/G: An Architecture for a Resource Management and Scheduling in a Global Computational Grid," *4th International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*, Beijing, China, IEEE Computer Society, Los Alamitos, CA, May 2000, pp. 283–289.

[50] http://www.csse.monash.edu.au.

[51] WebFlow, http://www.npac.syr.edu/users/haupt/WebFlow.

[52] Allen, G., E. Seidel and J. Shalf, "Scientific Computing on the Grid," *Byte*, Spring 2002, pp. 24–32.

[53] Cactus Code, http://www.cactuscode.org.

[54] Fox, G. C., "Portals and Frameworks for Web Based Education and Computational Science," http://www.new-npac.org/users/fox/documents/pajavaapril00.

[55] http://hotpage.npaci.edu.

[56] http://www.ascportal.org.

[57] Condor, http://www.cs.wisc.edu/condor/condorg.

CHAPTER 3

Building Up a Grid

3.1 Introduction

In this chapter we give practical guidelines to building up a grid and to using its facilities. For the sake of clarity, the chapter refers to a simple yet realistic grid, as shown in Figure 3.1.

The grid represented in the figure is composed of four machines interconnected by a transfer control protocol/Internet protocol (TCP/IP) network. Each machine is identified by its FQDN (i.e., the combination of its hostname plus its domain name). The four FQDNs are: "picasso.elemgrid.org," "matisse.elemgrid.org," "mozart.unile.it," and "bach.unile.it." The commands and the installation process are relative to the version 2.2 of the GT, and it is assumed that the machines are equipped with the RedHat Linux version 7.2 operating system, so each command is referred to this operating system. More on the Linux operating system and its commands can be found in Appendix A.

Though the description of the GT installation procedure might be more or less substantially adapted to different versions of the tool (and of the operating system), we try here to evince the main conceptual steps of the procedure and their general aspects, rather than proposing a simple "recipe" stuck to a specific code and system version. Consequently, we hope that the conceptual relevance of the proposed discussion remains valid even when different versions of the toolkit or of the Linux operating system are considered, thus offering the reader a useful and nonevanescent guide to build up a grid.

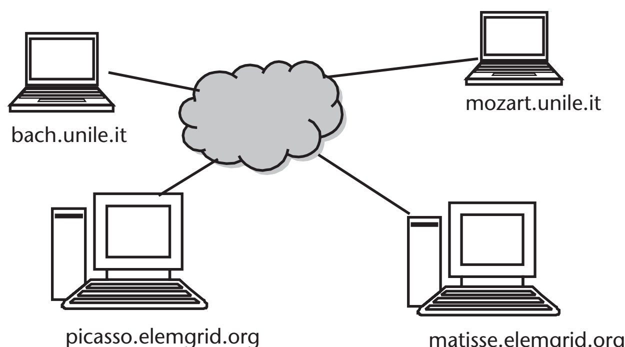

Figure 3.1 A sample grid, made up of four client/server machines. Each machine is connected with the network and identified by its unique fully qualified domain name (FQDN).

In a grid, the system administrator can freely assign the role (client, server, or both) each machine can play. A server machine is a machine whose resources are deployable by grid users, while client machines are those machines from which users can launch Globus commands to access grid resources. On client machines, users develop grid-enabled applications as well, making use of Globus software development kit (SDK). In our example, we suppose for simplicity that each machine runs both as server and as client platform. Consequently, users can develop grid-enabled applications on any machine in the grid and launch the running of the application on any machine in the grid as well.

GT should preferably be installed on a shared file system. In our book, we assume that our machines do not share a file system; therefore, the installation steps must be repeated for each machine.

3.2 Recalling Globus Basic Concepts

In order to make this chapter self contained, we now shortly recall the Globus main components, with specific attention paid to the role played in our configuration. The basic concepts on Globus are described in Chapter 2, and we refer the reader there for further details.

GT is described by its authors as being made up of three pillars [1, 2]:

- The RM pillar is responsible for scheduling and allocating resources specifying, for example, resource requirements and the operations to be performed, such as process creation or data access.

  Core RM services are managed by the GRAM. Each GRAM is responsible for a set of resources operating under the same allocation policy, often implemented by a local RM system. Thus, a computational grid built with Globus typically contains many GRAMs, each responsible for a *local* set of resources. For the sake of simplicity, in our example we just consider the UNIX *fork* system call as a local RM (i.e., we do not suppose any aggregation of resources via distributed management technologies other than Globus). In other terms, each machine is equipped with its own GRAM talking directly with the local operating system.

  GRAMs can interpret application requests via a standard API and a specific language (RSL). This language has a simple syntax that allows users to specify job requests and their characteristics (e.g., number of processes to be launched, working directory, and threading).

  The user interface to GRAM is the *gatekeeper*. When a job is submitted, the request is sent to the gatekeeper of the remote computer. The executable, the standard input and output, as well as the name and port of the remote computer, are specified as part of the job request (via RSL). The job request is handled by the gatekeeper, which creates a job manager for the new job. The job manager handles the execution of the job, as well as any communications with the user. It starts and monitors the remote program, communicating changes of status back to the user on the local machine. When the remote application terminates, correctly or with a failure, the job manager terminates as well. A

GRAM gatekeeper must be running on each server computer to run jobs remotely.

Another relevant component of the RM pillar is the GASS component, a service implementing a variety of automatic and programmer-managed data-movement and data-access strategies, enabling programs running on remote locations to read and write local data. In this example, our machines will act also as GASS server.

- The IS pillar collects information about the structure and state of resources (e.g., their current load and usage policy).

  IS relies on the LDAP. LDAP is a *directory service* (i.e., it allows storing and retrieving resources via a network protocol, as domain naming system (DNS) does for machines interfaced with the Internet). LDAP is a platform independent open protocol. Via LDAP, the IS pillar permits an effective monitoring of distributed resources. Globus information management infrastructure is built on the top of a widespread LDAP package, the open source OpenLDAP package, which is included inside Globus software.

  The MDS is the IS core. MDS has a distributed architecture. It is basically composed of GRIS and GIIS. Each GRIS provides information about the status of resources available in each node. Each GIIS gathers data from multiple GRIS resources. In our example, we consider the default Globus installation, which installs a GRIS on each server machine (see Figure 3.2).

- The DM pillar contains an extended version of the ftp, GridFTP, including features like partial file access and management of parallelism for high-speed transfer.

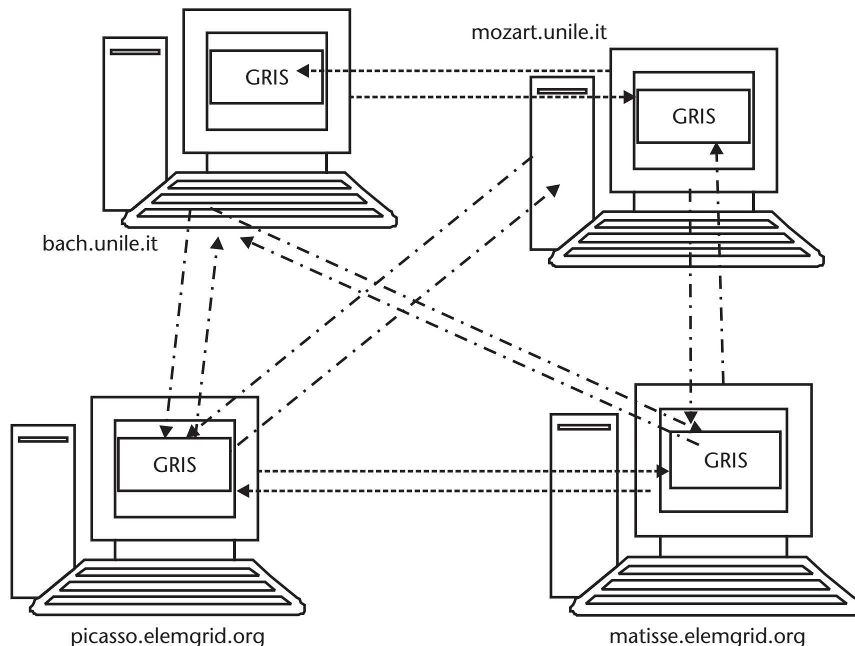

Figure 3.2 Client machines can query GRIS installed on each server machine.

- GSI ensures fundamental security services such as authentication, confidentiality, and integrity. More specifically, it guarantees *mutual authentication* among parties, relying on SSL protocol. SSL is based on public-key cryptography and assumes that a trusted neutral entity, the CA, exists and guarantees the authenticity of keys. Globus security infrastructure is built on the top of a widespread SSL package, the open source Openssl package, which is included into the Globus software.

Once the basic concepts of Globus have been recalled, we are ready to describe the set up of a real grid.

3.3 Setting Up the Environment

3.3.1 Hardware Requirements

Globus supports platforms with all UNIX and Linux flavors. The toolkit is not CPU intensive, nor is it memory intensive, so CPU and memory requirements will depend on the role of hosts in the grid. Hosts acting as *gateways* to other resources can be thin hosts, while hosts designed to provide computing services for grid jobs should have enough computing power and memory to sustain the computational requirements of the jobs targeted to them.

In conclusion, there are no specific hardware requirements: each machine can be a node in a grid, provided that it is assigned suitable roles and tasks.

3.3.2 Software Requirements

Of course, GT is required. It includes Globus Packaging Toolkit (GPT), which allows a totally automatic installation of GT. Furthermore, grid machines should be synchronized. Time synchronization is important for authentication. When users attempt to authenticate, they must present a proxy that has a timestamp and duration associated with it. If this proxy is presented to a host that is not time synchronized with the host on which the proxy was created, the users may not be able to authenticate with GT services. The timestamp of the proxy may be later than the current time on the host to which the proxy is being presented. A widespread time synchronization toolkit is Network Time Protocol (NTP). NTP is freely downloadable from the site www.ntp.org.

In conclusion, two basic packages are needed:

1. GT;
2. NTP.

3.3.3 Setting Up the Network

The grid is designed to connect devices with the TCP/IP protocol stack. The machines must be interfaced with each other via the TCP/IP network (it can be an isolated LAN or WAN as well). An FQDN is needed for each machine. FQDN is the address of a system, consisting of its hostname and its domain name (e.g., in our grid, hostnames are “matisse,” “picasso,” “bach,” and “mozart,” while domains

are "elemgrid.org" and "unile.it," and the FQDNs are "matisse.elemgrid.org," "picasso.elemgrid.org," "mozart.unile.it," and "bach.unile.it").

In conclusion, the standard procedures for creating a TCP/IP-based computer network must be performed.

3.3.4 Before Installing Globus

The Globus team suggests creating a separate user identifier (userid), such as "globus," under which GT daemons will run (this simplifies debugging). As some commands must be run by root, we assume that the whole installation will be performed from the root userid.

Before starting the installation, two directories should be created:

- A directory on which all GT software is installed, in our example:

"*/usr/local/globus*"

- A directory on which GPT is installed, in our example:

"*/usr/local/gpt*"

To communicate these directories to the Globus installation software, two environment variables (see Appendix A for details on environment variables) should be set:

1. The environment variable named "GPT_LOCATION" must point to the GPT installation directory.
2. The environment variable named "GLOBUS_LOCATION" must point to the Globus installation directory.

The commands to set these variables depend on the used shell:

- In case of shell belonging to the C family, they are:

```
setenv GPT_LOCATION   /usr/local/gpt
setenv GLOBUS_LOCATION        /usr/local/globus
```

- In case of Bourne shell:

```
export GPT_LOCATION=/usr/local/gpt
export GLOBUS_LOCATION=/usr/local/globus
```

As also suggested in Appendix A, these commands should be inserted into the *profile* script of the used userid (root in our example) and of each user of the grid (as explained later).

From now on, we refer to values assumed by the two variables with the usual form $GPT_LOCATION and $GLOBUS_LOCATION, in accordance with the most common notation adopted by computational scientists.

3.4 Globus Installation

After setting up the environment, the next step is the installation of GT. Of course, the first problem is finding the package. The full package is available in the attached CD-ROM, but we now describe the standard procedure to download it via the Internet.

3.4.1 Downloading the Package

GT services are given in a number of freely downloadable *bundles* [3]. A bundle is a collection of packages that can be installed and built with the GPT, also freely downloadable from the Globus site. The bundles reflect the GT structure. Each pillar is associated with three bundles: client, server, and SDK. Client bundles have to be installed on client machines (i.e., the machines launching applications). Server bundles refer to server machines (i.e., machines where applications run). SDK bundles should be installed when the goal is to develop a grid-enabled application that makes use of the Globus API.

There are source and binary bundles available. Installing precompiled binaries helps save the storage space required by the code, and it skips the compilation phase of the installation. The source distribution is preferred when the user intends to make changes to the GT code or debug the GT code at the source level or if she needs to install the GT on a system for which precompiled binaries are not available.

Anyway, when possible, we suggest installing source bundles: in a good number of cases (as further described in the book), they are strongly required. For this reason, from now on we refer to the *source* distribution.

Before installing Globus software, the GPT should be downloaded and installed. Suppose we download (via ftp or HTTP) the compressed archive named "gpt-###.tar.gz" (where ### generally contains the version number).

To extract the GPT files from the archive:

```
gunzip -c gpt-###.tar.gz -dc | tar xf -
```

This command creates a directory named "gpt-###" with GPT files. Move to that directory:

```
cd gpt-###
```

Build the package:

```
./build_gpt
```

If the environment variable GPT_LOCATION has been set, the GPT software is now installed in $GPT_LOCATION directory.

Now the Globus bundles can be downloaded (possibly in a directory other than the $GLOBUS_LOCATION) and installed.

3.4.2 Installing the Toolkit

The bundles are distributed in a compressed archive form, and files *must not be extracted* from it. The following command must be issued:

```
$GPT_LOCATION/sbin/gpt-build -verbose <bundle> <flavors>
```

Where <*bundle*> should be replaced with the bundle name and <*flavors*> with a string expressing compile options. This lets the user select his favorite compiler, architecture (32 or 64 bit), use of debugging, and use of thread. The Globus site contains a table of how to build the different packages, as reported in Table 3.1. The suggested flavors are for a 32-bit architecture with debugging turned on, always using the *gcc* compiler, and using threads when threading is applicable.

So the system administrator should type the following commands:

```
$GPT_LOCATION/sbin/gpt-build -verbose globus-data-management-
server-###.tar.gz gcc32dbg
$GPT_LOCATION/sbin/gpt-build -verboseglobus-information-services-
-server.-###tar.gz gcc32dbgpthr
$GPT_LOCATION/sbin/gpt-build -verbose globus-resource-
management-server-###.tar.gz gcc32dbg
$GPT_LOCATION/sbin/gpt-build -verbose globus-data-management-
client-###.tar.gz gcc32dbg
$GPT_LOCATION/sbin/gpt-build -verbose globus-information-
services-client-###.tar.gz gcc32dbgpthr
$GPT_LOCATION/sbin/gpt-build -verbose globus-resource-
management-client-###.tar.gz gcc32dbg
$GPT_LOCATION/sbin/gpt-build -verbose globus-data-management-
sdk-###.tar.gz gcc32dbg
$GPT_LOCATION/sbin/gpt-build -verbose globus-information-
services-sdk-###.tar.gz gcc32dbgpthr
$GPT_LOCATION/sbin/gpt-build -verbose globus-resource-
management-sdk-###.tar.gz gcc32dbg
```

where ### contains the version number.

Once all of the bundles are installed, run the configuration scripts that will customize the Toolkit installation to the current host, by launching the command:

```
$GPT_LOCATION/sbin/gpt-postinstall
```

Run the Globus script that installs GSI files into the "/etc/grid-security" directory:

Table 3.1 Flavor Names as Suggested for the Standard Case

Bundle	*Flavors*
DM	gcc32dbg
IS	gcc32dbgpthr
RM	gcc32dbg

```
$GLOBUS_LOCATION/setup/globus/setup-gsi
```

There are two environment scripts called "$GLOBUS_LOCATION/etc/globus-user-env.sh" and "$GLOBUS_LOCATION/etc/globus-user-env.csh." You should read in whichever one corresponds to the type of shell you are using.

For example, if you are using a shell belonging to the C family, you must run:

```
source $GLOBUS_LOCATION/etc/globus-user-env.csh
```

In case of the Bourne family, you must type:

```
.$GLOBUS_LOCATION/etc/globus-user-env.sh
```

At this point GT is installed in the $GLOBUS_LOCATION directory (see Table 3.2. Rooting at this directory, a number of Globus folders have been automatically created:

Another relevant folder is:

"/etc/grid-security"

This contains Globus information related to security, as explained in detail in the following section.

3.5 Globus Configuration

Once the bundles have been installed, the configuration of the grid mostly focuses on security aspects, with *authorization* and *authenticity* being the most critical ones. Both services are managed through the univocal identification of entities in the grid via the distinguished name and through the use of certificates [4], signed by a trusted CA, to be exhibited upon request (see Figure 3.3).

With regard to *authorization*, GT leaves the system administrator the freedom to select access to resources for users by creating user accounts on the server machines. As already explained in Chapter 2, users are given a *globally unique* name (a DN), which identifies them in the context of the grid. Then, the system

Table 3.2 List of GT Directories

$GLOBUS_LOCATION/etc/	It contains some relevant configuration files. The most relevant is globus-gatekeeper.conf, containing a number of configuration information, such as the gatekeeper port number and the location of the security data
$GLOBUS_LOCATION/bin/	With Globus commands inside
$GLOBUS_LOCATION/man/	With Globus man pages inside
$GLOBUS_LOCATION/lib/	With Globus libraries inside
$GLOBUS_LOCATION/include/	With Globus include files inside
$GLOBUS_LOCATION/var/	Where the GRAM gatekeeper logs its activity in a file called globus-gatekeeper.log

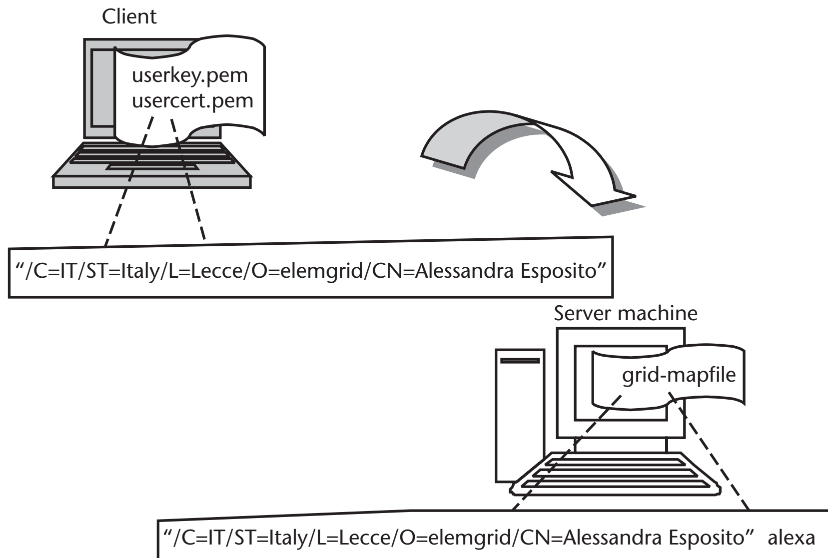

Figure 3.3 In order to allow users to access grid services, client and server machines must be configured in a suitable fashion. On the client side, users should be given an account, a DN, and a certificate associated with that DN. On the server side, the core configuration step is the generation of the grid-mapfile file, where the global DN is mapped to a local userid.

administrator must define a match between users (identified by their DN) and accounts on server machines. When a candidate user requests access to resources located on a server machine, he exhibits his certificate, with his DN inside. GT reads the DN and checks whether the user is permitted to use the requested resources.

With regard to the *authentication* service, GT adopts a *mutual* authentication, where both parts aiming at communicating must prove to one another their identity. Because of this, every user and service must own a certificate. Users connect to client machines where their certificate is installed and from those machines, they contact remote server machines by entering Globus commands. The authentication handshake completes successfully if:

1. Both user and server own a certificate;
2. Globus security procedures recognize each certificate as valid.

3.5.1 Authorization

A file named "grid-mapfile," must be created in the "/etc/grid-security" folder of server machines, to specify the list of authorized users of resources. This file contains the information needed by the Globus gatekeeper to map a request for GT services from a user with a particular global subject name (DN) to a local user login name on that system. The file contains records consisting of user distinguished names and the local login name or account that should be used on that system. If a match exists, then the requested GT service is provided and will be invoked under the appropriate user login name or account, as determined in "grid-mapfile." Each entry of the "grid-mapfile" is a couplet of quoted credential name (the subject of a certificate) and an unquoted local user name. An example of "grid-mapfile" is the following:

"/C=IT/ST=Italy/L=Lecce/O=elemgrid/CN=AlessandraEsposito" alexa

"/C=IT/ST=Italy/L=Lecce/O=elemgrid/CN=LucianoTarricone" luciano

In this way, we ask to match two remote users, identified by their subject names, respectively, with the local user name "alexa" and "luciano."

3.5.2 Authentication

The authentication service is based on the SSL protocol. Each user and service in the grid must be identified by a certificate containing the public cryptographic key and the signature of the CA. In GT, certificates can be created only by the CA, who reviews the certificate request submitted by the user and accepts or denies it according to an established policy.

If you already have a CA in your network, you can use it for the certification in the Globus context. Otherwise, there are two possibilities:

1. You can request certificates to the Globus CA. The Globus team, in fact, offers the opportunity to contact its own CA via e-mail. The toolkit contains a useful script to generate a certificate request to be sent to the Globus CA.
2. You can create your own CA. This possibility is very useful if you have an isolated network that does not communicate with the outside world or if you are just experimenting with a fictitious network. In these cases, you must first create the CA certificate and then user and service certificates. The Openssl package contains all of the commands necessary to achieve this goal, though GT is equipped with a very useful script simplifying this task.

These two alternatives are described next. For each alternative, we explain how to create:

1. *User certificates*. Each user of the grid must be authenticated. To do that, a certificate must be created and distributed in the grid.
2. *Gatekeeper certificates*. When a job is submitted to the gatekeeper by the client, a process of mutual authentication occurs. On one side it ensures that the client has permission to execute jobs on the computer. On the other side, it checks that the gatekeeper is the correct resource. For this reason, the gatekeeper must own a certificate and key.

3.5.3 Using the Globus CA

3.5.3.1 User Certificate

GT includes a certificate request generation script to create keys and certificate requests. The user must enter the command:

```
grid-cert-request
```

It is important that the user is running from its normal user account, not "root" or "globus." The CA will not sign certificates for the accounts "root" or "globus,"

because they are local accounts, not necessarily affiliated with a real person. This command asks for a password to protect the user's private key. The command also asks for a subject name, which should be something like

"/O=Grid/O=Globus/OU=elemgrid.org/CN=AlessandraEsposito"

where OU must match with the host DNS name and CN with the user name as returned from the finger command.

When the user runs this command, it generates three files:

1. The first file is a certificate request named "usercert_request.pem." The user's public key is inserted into the certificate request.
2. The second is the user's private key, saved on a separate file named "userkey.pem," which is encrypted using the user's password.
3. The third is a 0-byte file, named "usercert.pem." It is merely a placeholder that serves as a reminder where to put the certificate when the CA responds to the request.

The request must be emailed to the CA (ca@globus.org for using Globus CA), which, upon receiving the request, reviews it and signs it electronically.

When the response arrives, the user should create a directory inside his home directory, name it ".globus," and save there the e-mail onto a file named "user cert.pem." In the same directory the user should save the "userkey.pem" file, too.

In the end, the user will have a "userkey.pem" and "usercert.pem" in its ".globus" directory.

3.5.3.2 Host Certificate

Host certificate and key are requested and created in a similar manner by the system administrator. The certificate and key of the gatekeeper, however, do not require a password.

The following command should be run as root to get a gatekeeper certificate, replacing the text <hostname> with the fully qualified hostname:

```
grid-cert-request -service host -host hostname
```

For example, if you are requesting a certificate for the gatekeeper running on "matisse.elemgrid.org," you should type:

```
grid-cert-request -service host -host matisse.elemgrid.org
```

This command generates the gatekeeper certificate request and the gatekeeper private key. The certificate request should then be e-mailed to the CA. When the certificate arrives, the contents of the e-mail should be saved in:

/etc/grid-security/hostcert.pem

The private key should be saved in:

/etc/grid-security/hostkey.pem

These files should be owned by root with permissions 600 (i.e., they should be readable and writeable only by root), so the following command should be run as root:

```
chmod 600 /etc/grid-security/hostcert.pem

chmod 600 /etc/grid-security/hostkey.pem
```

3.5.4 Using a Local CA

To create a local autonomous CA, three steps must be performed:

- Generation of CA certificate;
- Installation of the CA certificate;
- CA configuration.

Generation To generate a CA certificate, a useful script named "CA.sh" located in the directory named "$GLOBUS_LOCATION/bin" can be used. The "CA.sh" script uses the configuration file named "openssl.cnf," included in the GT software, and expects it to be located in the installation directory of Globus, namely $GLOBUS_LOCATION. In some GT versions, the file must be manually copied into that folder; otherwise, the script returns an error:

```
cp /usr/share/ssl/openssl.cnf $GLOBUS_LOCATION
```

Then, the following command can be typed:

```
$GLOBUS_LOCATION/bin/CA.sh  -newca
```

The command asks for the CA password to protect the CA private key. Then, it asks for the CA DN, requesting the following fields, some of which are not compulsory:

- Country code (C);
- State name (ST);
- Locality name (L);
- Organization name (O);
- Organization unit name (OU);
- Common name (CN);
- E-mail address (Email).

Once these fields have been inserted, the resulting sequence is the DN of the CA. Suppose we input the strings, as listed here:

- Country code: IT;

- State name: Italy;
- Locality name: Lecce;
- Organization name: elemgrid;
- Common name: CA

With these reported choices, the DN looks like:

"/C=IT/ST=Italy/L=Lecce/O=elemgrid/CN=CA"

As can be verified by issuing the command:

```
openssl x509 –in ./demoCA/cacert.pem –noout –issuer
```

Where "cacert.pem" is the name of the file containing the CA certificate, generated by the CA.sh script. Meanwhile the encrypted private key is saved in a file named "cakey.pem." The CA.sh script creates a directory named "demoCA," where the CA maintains a database of the certificates it manages. In this directory, the script puts the file named "cacert.pem," while the file named "cakey.pem" is put into the "demoCA/private/" folder.

Installation Once the certificate and the key have been generated, CA's certificate needs to be distributed to all server and client machines. It must also be installed there with the appropriate name and in the suitable directory.

GT requires that the certificate is named with the hash value of its subject name (i.e., it must have a filename like "*hash_value.0,*" where *hash_value* is the hash value of the subject name of the certificate). This is used in Openssl to form an index to allow certificates in a directory to be looked up by subject name.

Suppose now that the current directory is that created by the "CA.sh" script and named "demoCA." To calculate the hash value, the following command should be run:

```
openssl x509 –in cacert.pem –noout –hash
```

Suppose now that the command returns the hash value *b37bb35*. The following command should be run in order to install the certificate in the final directory:

```
cp cacert.pem /etc/grid-security/certificates/b37bb35.0
```

If you prefer, you can rename it:

```
mv cacert.pem /etc/grid-security/certificates/b37bb35.0
```

or create a symbolic link:

```
ln –s cacert.pem /etc/grid-security/certificates/b37bb35.0
```

Configuration Once the CA certificate has been installed in the final directory, a configuration file named "*hash_value.signing_policy*" ("*b37bb35.signing_policy*" in the example) must be created and installed in the same directory as the certificate.

The file describes the security policy for the grid. The policy is described by the extended access control lists (EACL). EACL enumerate CAs allowed access to grid objects and the type of access they are granted. Their contents look like:

```
#token type    |    def.authority    |    value
# EACL entry #1
access_id_CA       x509                 '/C=IT/ST=Italy/L=Lecce/O=elemgrid/CN=CA'
pos_rights         globus               sign
cond_subjects      globus               '"/C=IT/ST=Italy/*"'
#end of EACL
```

The character '#' at the beginning of a line marks a comment line. In the line beginning with the token type "access_id_CA," the CA is identified (i.e., the DN of the CA must be defined). In the line beginning with the token type "pos_rights," it is declared *what* the CA can do (i.e., sign certificates). In the line beginning with the token type "cond_subjects," we say *whom* the CA accepts.

In this example, the file says that:

1. Line with the *access_id_CA* token—the CA is identified by the *x509* subject name:

   "C=IT/ST=Italy/L=Lecce/O=elemgrid/CN=CA"

2. Line with the *pos_rights* token—the CA can *sign* certificates for the *globus* community.
3. Line with the *cond_subjects* token—the CA accepts only certificates with a subject name having the fields C and ST equal to "IT" and "Italy," respectively.

3.5.4.1 User Certificate

Suppose now that the current directory is the parent directory of "demoCA." To create the certificate request and the private user key, the following command can be used:

```
openssl req –new –keyout userkey.pem –out userreq.pem
```

This command will ask for a password to protect the user private key. The command also will ask for a subject name, which should look something like

"/C=IT/ST=Italy/L=Lecce/O=elemgrid/CN=AlessandraEsposito"

where CN must match with the user name. This key generates the user private key ("userkey.pem") and the certificate request ("userreq.pem").

To request the local CA to sign the certificate, the following command can be used:

```
openssl ca –policy policy_anything –out usercert.pem –infiles
userreq.pem
```

This command will ask for the CA password to decrypt its private key. This step outputs the user signed certificate named "usercert.pem."

To copy the certificate in the suitable directory, the following commands can be used:

```
cp userkey.pem $HOME/.globus/
cp usercert.pem $HOME/.globus/
```

Where $HOME points to the user home directory.

To set the permissions:

```
chmod 600 $HOME/.globus/userkey.pem
chmod 600 $HOME/.globus/usercect.pem
```

To verify the subject name, the following command can be used:

```
openssl x509 –in usercert.pem –noout –subject
```

3.5.4.2 Host Certificate

To create the gatekeeper certificate request and the private key, the following command must be typed:

```
openssl req -nodes –new –keyout hostkey.pem –out hostreq.pem
```

where the option "–nodes" is needed to create a no-password private key.

The command asks for a subject name, which should look something like:

"/C=IT/ST=Italy/L=Lecce/O=elemgrid/CN=picasso.elemgrid.org"

where CN must match with the hostname. This key generates the host private key ("hostkey.pem") and the certificate request ("hostreq.pem").

To request the certificate to the local CA the command is:

```
openssl ca –policy policy_anything –out hostcert.pem –infiles
hostreq.pem
```

This command asks for the CA password to decrypt its private key. This step outputs the host signed certificate named "hostcert.pem."

To copy the certificate in the suitable directory:

```
cp hostkey.pem /etc/grid-security/
cp hostcert.pem /etc/grid-security/
```

To set the permissions:

```
cd /etc/grid-security/
chmod 600 hostkey.pem
chmod 600 hostcert.pem
```

3.6 Services Start Up

From the grid user and administrator point of view, it is extremely useful that all of the services are automatically started during the bootstrap of each machine in the grid. This requires some configuration steps, which are schematically described in this section. What follows can be dependent on the operating system and might need some small customizations.

3.6.1 Resource Management

A server machine must have a GRAM gatekeeper active. Red Hat Linux includes the eXtended InterNET services daemon (called "*xinetd*"), responsible for starting programs that provide Internet services. The xinetd configuration file ("/etc/services") lists services to be started by xinetd. In order to include the GRAM gatekeeper in this list, edit this file by adding the line:

gsigatekeeper 2119/tcp

This line asks the Linux xinetd daemon to start at port 2119 a service employing the tcp protocol and named gsigatekeeper.

Next, create a file in the "/etc/xinetd.d/" folder, name it "globus-gatekeeper" and add the following:

```
service gsigatekeeper
{
socket_type    =    stream
protocol       =    tcp
wait           =    no
user           =    root
env            =    LD_LIBRARY_PATH=GLOBUS_LOCATION/lib
server         =    GLOBUS_LOCATION/sbin/globus-gatekeeper
server_args    =    -conf GLOBUS_LOCATION/etc/globus-gatekeeper.conf
disable        =    no
}              =
```

where GLOBUS_LOCATION must be replaced with the globus installation folder ("/usr/local/globus" in our example).

This file lists the characteristics of the service to be started by xinetd. Details on the single commands in the file can be found in Linux manuals.

Once the configuration files have been updated, restart xinetd:

```
/etc/rc.d/init.d/xinetd restart
```

3.6.2 Information Services

As discussed in Chapter 2 and shortly resumed at the beginning of the present chapter, MDS is the core of the processes collecting information from grid nodes.

During the installation phase, a script named “SXXgris” is automatically installed in the directory named $GLOBUS_LOCATION/sbin. This script must be used both to start MDS manually:

```
$GLOBUS_LOCATION/sbin/SXXgris start
```

and to stop it

```
$GLOBUS_LOCATION/sbin/SXXgris stop
```

To start MDS automatically, add the following line to the file named “/etc/rc.local”:

```
$GLOBUS_LOCATION/sbin/SXXgris start
```

where GLOBUS_LOCATION must be replaced with the globus installation folder (“/usr/local/globus” in our example).

3.6.3 Data Management

Data transfer in Globus is performed by using GridFTP (see Chapter 2 and the beginning of the current chapter).

To start automatically the GridFTP service, add the following line to the file named “/etc/services”:

```
gsiftp    2811/tcp
```

This line asks the Linux xinetd daemon to start a service named “gsiftp” at port 2811. Then, create the file named “grid-ftp" in the “/etc/xinetd.d/” folder with the following contents:

```
service gsiftp
{
socket_type     =     stream
protocol        =     tcp
wait            =     no
user            =     root
env             =     LD_LIBRARY_PATH=GLOBUS_LOCATION/lib
server          =     GLOBUS_LOCATION/sbin/in.ftpd
server_args     =     -l -a
disable               no
}
```

where GLOBUS_LOCATION must be replaced with the Globus installation folder (“/usr/local/globus” in our example).

Details on the commands contained in the file can be found in Linux manuals.

Finally, xinetd must be restarted by issuing the command:

```
/etc/rc.d/init.d/xinetd restart
```

3.7 Introducing a New User to the Grid

Once the basic infrastructure of the grid has been activated, users must be registered in accordance with the security issues reported in Section 3.5. This requires some operations both on the server and on the client side.

3.7.1 Client Side

On the client side, the following procedure is suggested:

1. Create an account on a Globus-enabled client machine (i.e., a machine where client bundles have been installed).
2. Login to that machine with the account created in step 1.
3. Set up your environment and path; if you are using a C shell, you must add the following lines to your profile:

   setenv GLOBUS_LOCATION /usr/local/globus

   source $GLOBUS_LOCATION/etc/globus-user-env.csh

   In the case of the Bourne shell, you must insert the following lines in your profile:

   export GLOBUS_LOCATION=/usr/local/globus

   . $GLOBUS_LOCATION/etc/globus-user-env.sh

4. Create a user certificate, as reported in Section 3.5.1.

3.7.2 Server Side

On the server side, the following procedure is suggested:

1. Create an account on a server machine.
2. Install the user certificate generated on the client side.
3. Change the "grid-mapfile" file, as reported in Section 3.5.1.

3.8 Globus-Relevant Commands to Use the Grid

We are now ready to let users access all of the grid facilities and services. In this section, the main commands to use the grid and its resources are reported.

3.8.1 Authentication

Globus authentication requires the use of proxies, a convenient mechanism for reducing the number of times users must enter their password. If many jobs are to be submitted in a few hours, the tedium of reentering the password can be avoided by creating a proxy. The proxy is a certificate and a key, signed by the user, created using the *grid-proxy-init* program. The number of hours for which it will be valid, as well as the number of bits in the key, can be set by the user. The intention is that these proxies be used for short-term rather than long-term authentication. To create a proxy enter:

```
grid-proxy-init
```

The command requests the user password for decrypting its private key and outputs the proxy expiration date. By typing *grid-proxy-init*, the user obtains single sign-on capability for 12 hours (the default). To get rid of your proxy, enter:

```
grid-proxy-destroy
```

The proxy will self destruct after 12 hours (or whatever duration you set), but it is more secure to destroy it by yourself when your job is completed.

It is worth mentioning that authentication is needed in order to access every service in a grid. Therefore, the "grid-proxy-init" command must be typed before accessing any resources, as will probably be apparent after reading the beginning of the next subsection.

3.8.2 Resource Management

Running an executable on a remote machine is probably one of the most frequent needs in a grid. The executable to be launched can be resident on the targeted machine or not. Both cases can be managed. The suitable command for such task is:

```
globus-job-run
```

We recall that this command must be forwarded by the "grid-proxy-init" command.

When the executable to run is resident on the remote machine, the complete syntax of the command becomes:

```
globus-job-run host <executable> <arguments>
```

An example can be useful to explain it (see Figure 3.4).

Suppose we launch the following command from the machine named "bach.unile.it":

```
globus-job-run picasso.elemgrid.org /bin/ls /usr/local/bin
```

This command returns the list of files contained in the "/usr/local/bin/" folder of the machine called "picasso.elemgrid.org."

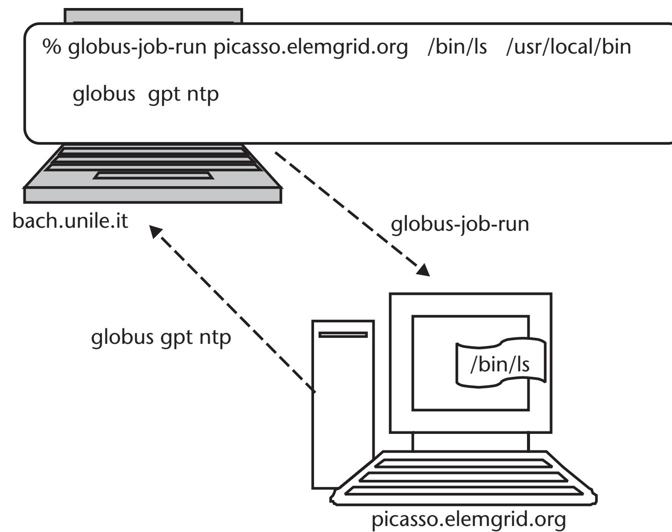

Figure 3.4 With the "globus-job-run" command, it is possibile to execute a remote program, resident on a server machine. On the client machine, "bach.unile.it", the user asks to execute the "ls" command resident on the server machine, namely "picasso.elemgrid.org". The command standard output is redirected to the client console.

Suppose now that you have developed an application, named "myapp," on the machine called "bach.unile.it." Suppose also that this application requires resources not available on machine "bach.unile.it" but available on other machines belonging to the grid ("picasso," for instance). The right thing to do is to run the application on the "picasso" machine. This can be done by simply using the "*–stage*" option of the globus-job-run command (see Figure 3.5).

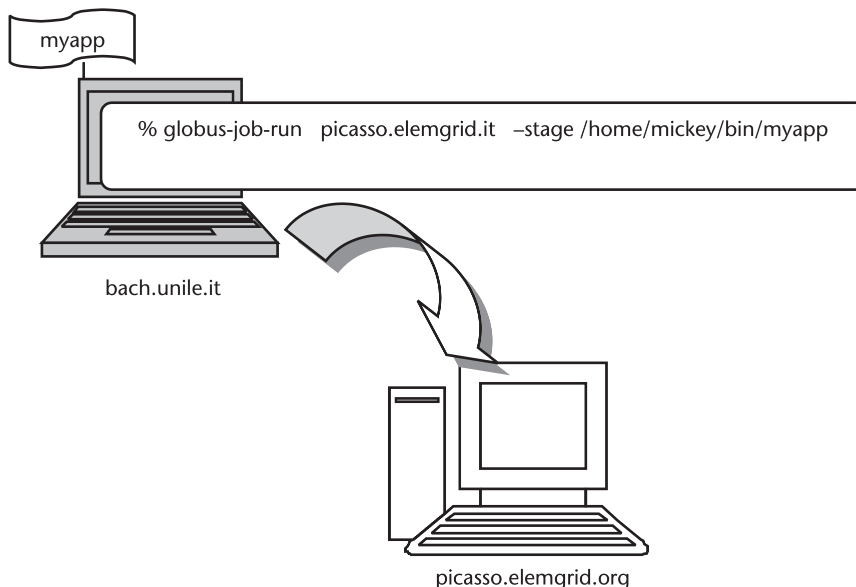

Figure 3.5 By giving the option "–stage" to the "globus-job-run" command, an executable resident on the local machine can be run in a remote fashion. The application is resident on the client machine, namely "bach.unile.it", and runs on the server machine, namely "picasso.elemgrid.org".

When this option is given, the command first stages the executable over the remote machine, then executes it, and finally removes the staged copy when the executable has finished.

In this case, the complete syntax becomes:

```
globus-job-run <hostname> –stage <executable> <arguments>
```

Returning to our example, in order to run the application named "myapp" remotely, the following command should be entered from the client machine (where the application resides):

```
globus-job-run picasso.elemgrid.org –stage /home/mickey/bin/myapp
```

If the application requires any input argument (e.g., an integer), it can be given with the same command:

```
globus-job-run picasso.elemgrid.org –stage
/home/mickey/bin/myapp 10
```

Consider now a more complex scenario (see Figure 3.6): suppose that the application "myapp" reads its input from a file named "myapp.in" (possibly generated by another application that has been launched from another client machine) and resident on the targeted machine, namely "picasso.elemgrid.org."

This case can be addressed by generating an RSL script file (RSL was introduced in Chapter 2):

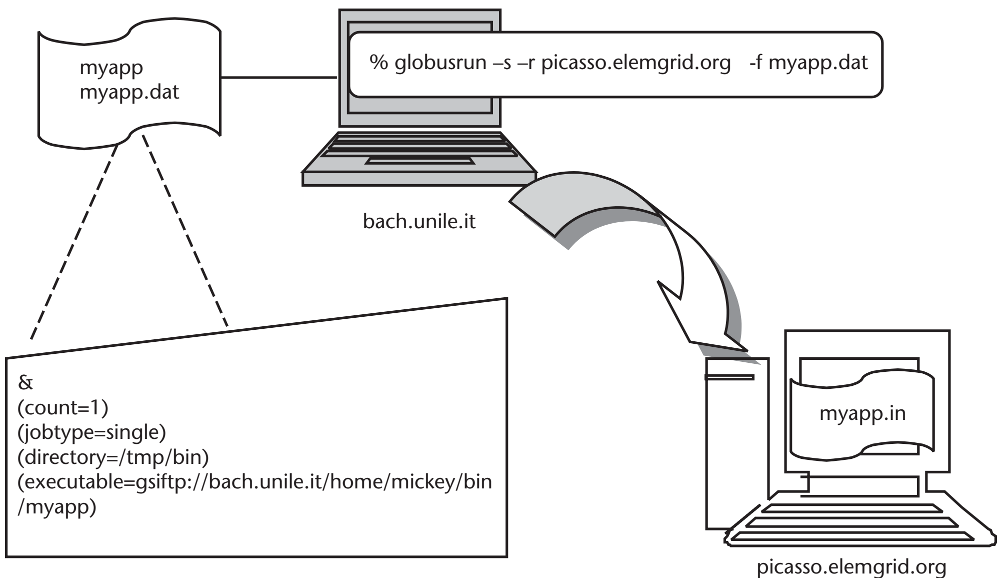

Figure 3.6 By writing RSL files, it is possibile to express the characteristics of the job to be launched. For example, it is possibile to ask that the application reads the input from a file located in the server machine. The application, namely myapp, and the RSL file, namely myapp.dat are resident on the client machine, namely "bach.unile.it," while the input file is located on the server machine. On the client machine, the command globusrun must be entered, passing to it the FQDN of the server machine and the filename of the RSL file.

```
&
(count=1)
(jobtype=single)
(directory=/tmp/bin)
(executable=gsiftp://bach.unile.it/home/mickey/bin/myapp)
(stdin=myapp.in)
```

This file says that the executable named:

"/home/mickey/bin/myapp"

and located in the directory named "/home/mickey/bin" of "bach.unile.it" machine (i.e., the client machine), should run in a not-threaded way (jobtype=single), with one active process (count=1), in the active directory called "/tmp/bin," and its input should be read from a file called "myapp.in."

This file is created on the machine where the application resides and is named "myapp.dat." You can then issue the command:

```
globusrun –s –r picasso.elemgrid.org –f myapp.dat
```

This command interprets the contents of the RSL file named "myapp.dat." So, in accordance with the directives contained in that file, it executes the program named "/home/mickey/bin/myapp" on the server machine named "picasso.elemgrid.org." The executable is launched in the "/tmp/bin" directory of the server machine and reads the input from a file named "myapp.in" and located in that directory.

If the application generates any output, it could be useful to save it into a file, named "myapp.out." To achieve this, it is sufficient to add the following line to the RSL "myapp.dat" file:

```
(stdout=myapp.out)
```

In this case, if the *globusrun* command is launched, at the end of the execution, the user can read the standard output by accessing the file "myapp.out" located in the "/tmp/bin" folder of the server machine. This can be done by using traditional ftp or by using Globus data-management programs, which will be discussed later. By the way, we suggest fully exploiting the potential of RSL, which allows users to recall files for standard input or standard output via an URL, as shown in Figure 3.7.

3.8.3 Information Services

Monitoring the status of facilities in the grid is extremely useful and important for end users. For instance, the identification of where a certain job is running can be better performed once the CPU occupation is checked for all of the nodes in the grid. The GRIS resources of Globus are extremely useful for such goals. An example is represented by the following command typed on a client machine (e.g., the "bach.unile.it" machine, as shown in Figure 3.8):

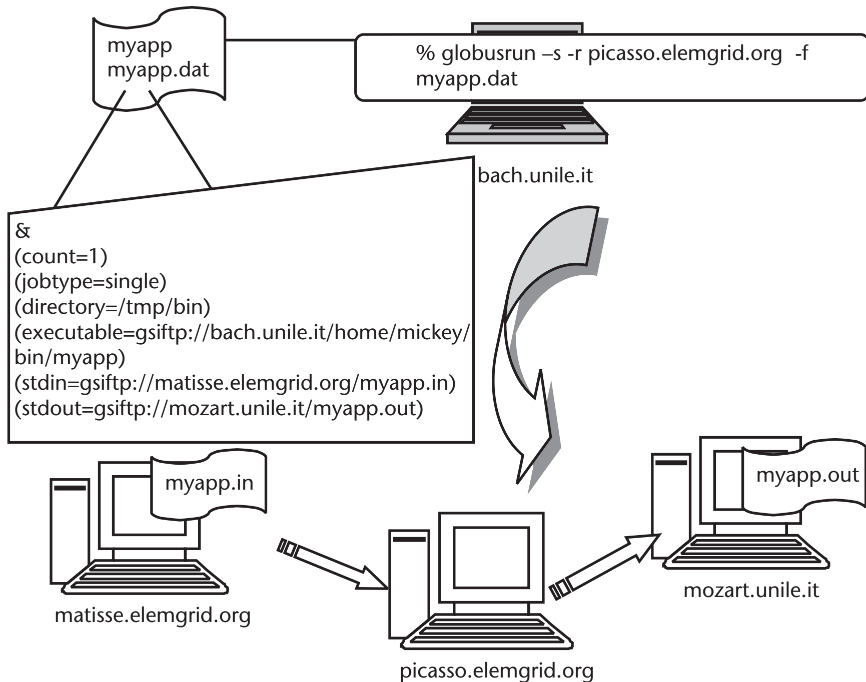

Figure 3.7 The RSL syntax allows users to ask that the application reads the input from or writes the output on a machine different from both the client machine and the hosting machine. In the example, the application, resident on "bach.unile.it" is launched on "picasso.elemgrid.org," reads the input from the file "myapp.in" located in the "matisse.elemgrid.org" machine, and produces its output by generating the file "myapp.out" in the "mozart.unile.it" machine.

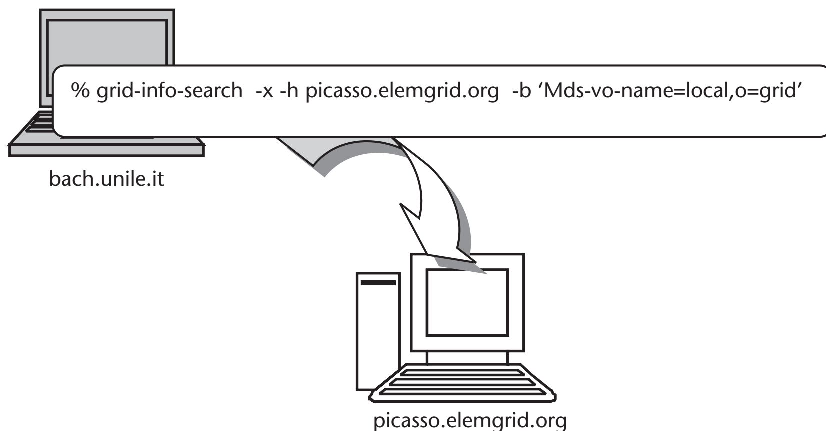

Figure 3.8 From a client machine, it is possibile to query remote GRIS about the status of resources located on remote machines. In the example, from "bach.unile.it," the user requests data with respect to all of the resources available on the remote machine named "picasso.elemgrid.org."

```
grid-info-search –x -h picasso.elemgrid.org –b 'Mds-vo-
-name=local,o=grid'
```

With this command, you ask information about all resources available on "picasso" ("-h" option) by querying the "picasso" local GRIS ("-b" option). The

option named "-x" is used to make an anonymous query (otherwise, an LDAP certificate must be generated and installed in a similar fashion as described for user and host certificates in Section 3.5.2)

Segments of the output returned by the above command are as follows:.

```
#
# filter: (objectclass=*)
# requesting ALL
#
# picasso.elemgrid.it,local,grid
dn:Mds-host-dn=picasso.elemgrid.org , Mds-vo-name=local,o=grid
objectClass=MdsComputer
objectClass=MdsComputerTotal
objectClass=MdsCpu
objectClass=MdsCpuCache
objectClass=MdsCpuFree
....
Mds-Computer-platform=I686
Mds-Computer-Total-nodecount=1
Mds-Cpu-Cache-12kb=512
...
```

The command returns the list of all resources available on the machine "picasso" (lines beginning with the token "objectClass") and, for each resource, gives their characteristics (lines beginning with the token Mds-).

3.8.4 DM

You can use GT to easily transfer files between machines. *Ftp* or secure copy (*scp*) are well-known protocols for such purposes. Unfortunately, many sites have deactivated ftp for security reasons, while scp is sometimes very slow because it has to encrypt all of the data to be transferred. With the Globus command "*globus-url-copy,*" you have an easy alternative.

Suppose you need to transfer the file named "myfile," located in the directory named "/home/alexa/" from machine "bach.unile.it" to "matisse.elemgrid.org". Login into the machine "bach.unile.it" and issue the following command:

```
globus-url-copy file:///home/alexa/myfile
gsiftp://matisse.elemgrid.org/home/alexa/myfile
```

Another way to transfer files, whose appeal will be evident when dealing with the GT API, in Section 3.9, is the use of a GASS server. Just look at the following example where we transfer a file from machine "bach.unile.it" to "matisse.elemgrid.org" (see Figure 3.9).

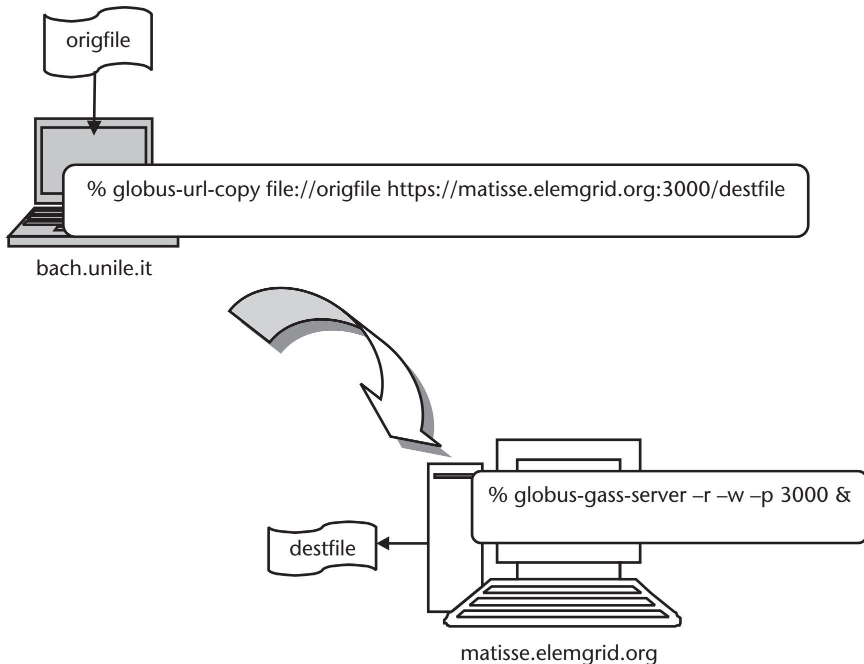

Figure 3.9 Globus procedure to transfer a file from a local machine to a remote machine. Before transferring the file with the globus-url-copy program, a GASS server must be active on the destination machine and the client has to be authenticated.

First, a GASS server must be started on the targeted machine. To do this, on "matisse.elemgrid.org," issue the *globus-gass-server* command:

```
globus-gass-server  -r -w  -p 3000 &
```

The command is started in background (by typing the "&" character at the end of the command), and the following options have been adopted:

- *-r* to enable read access to the local file system;
- *-w* to enable write access to the local file system;
- *-p port* to set the port number at which the GASS server listens.

The command returns the following line:

https://matisse.elemgrid.org:3000

As noticeable, the command returns an address, called base URL, which identifies the targeted machine. By default, this is an HTTP over SSL (HTTPS) URL, using the user's credentials. If the –i (insecure) option is set, the protocol used is HTTP (without security).

Next, on the bach.unile.it machine, authenticate yourself by launching the command:

```
grid-proxy-init
```

Finally, you can initiate the transfer. On “bach,” launch:

```
globus-url-copy file://origfile https://matisse.elem-
grid.org:3000/home/mickey/destfile
```

This command copies the file “origfile” from the current directory of “bach” onto the file “/home/mickey/destfile” of “matisse.” If simple customizations to the procedure described earlier are performed, it is even possible to request the transfer between a couple of remote machines, as shown in Figure 3.10.

3.9 Developing Grid-Enabled Applications

In the previous section, we described how a user can access grid facilities and services. Now, we focus on how a programmer can develop applications ready to run in a grid environment. To this purpose, a fundamental role is played by the Globus API (already described in Chapter 2). All of the Globus components offer an API written in language C, which allows calling Globus utilities in the context of an application. In this section, we introduce a very useful API, the GASS file access API. It allows reading or writing files on a remote machine.

Once a remote file is opened with the Globus calls, it can be read and written with ordinary UNIX I/O and C calls. Hence, a program can be modified to operate in a grid environment by simply using the Globus calls in place of open(), close(), fopen(), and fclose(), typical of C and UNIX.

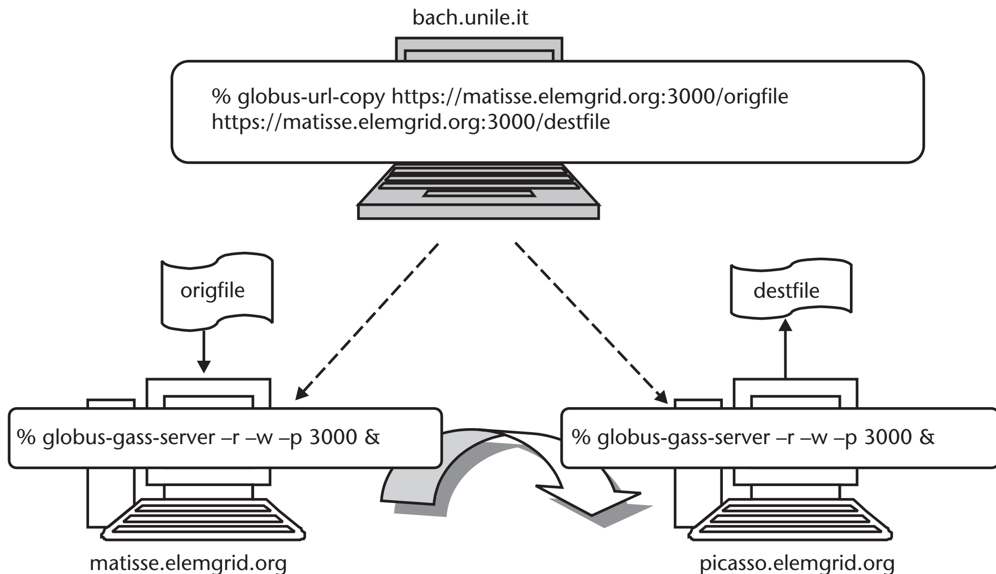

Figure 3.10 By typing the “globus-url-copy” command, it is possibile to ask the transfer of a file from a remote machine to another remote machine. In the figure, from “bach.unile.it”, the user asks to transfer the file named “origfile” from the “matisse.elemgrid.org” machine to the “picasso.elemgrid.org” machine. The procedure completes successfully if the GASS server has been previously activated on both the machines involved in the transfer (namely matisse and picasso) and if the user has been authenticated.

Suppose, for instance, that you have a code written in C language containing such lines:

```
fd = fopen("filename", ...);
fprintf(fd, ...);
fclose(fd);
```

You can migrate it towards a grid environment by changing it to:

```
fd = globus_gass_fopen("http://hostname:port/filename", ...);
fprintf(fd, ...);
globus_gass_fclose(fd);
```

The “globus_gass_fopen” function stages the file into the GASS cache, opens it locally, and returns the variable named “fd.” Then all of your I/O operations against that file are local. The “globus_gass_fclose()” function finally stages the file back to the URL.

Note that, before these functions are called, the following function must be called:

```
globus_module_activate(GLOBUS_GASS_FILE_MODULE);
```

This function returns GLOBUS_SUCCESS if GASS is successfully initialized, and you are therefore allowed to subsequently call GASS functions. Otherwise, an error code is returned, and GASS functions should not be subsequently called.

To deactivate GASS, the following function must be called:

```
globus_module_deactivate(GLOBUS_GASS_FILE_MODULE);
```

This function must be called once for each time GASS is activated.

3.9.1 An Example with Globus API

Suppose that you are implementing a routine on the “bach.unile.it” machine that needs to write on a remote file (see Figure 3.11).

Suppose that the file is located on the “picasso.elemgrid.org” machine, in the directory named “/home/alexa/” and that it is named “myfile.”

On “bach,” write down the following code:

```
#include <stdio.h>
#include "globus_common.h"
#include "globus_gass_file.h"
#include "globus_error.h"
main()
{
    FILE *fp;
    int rc=0;
```

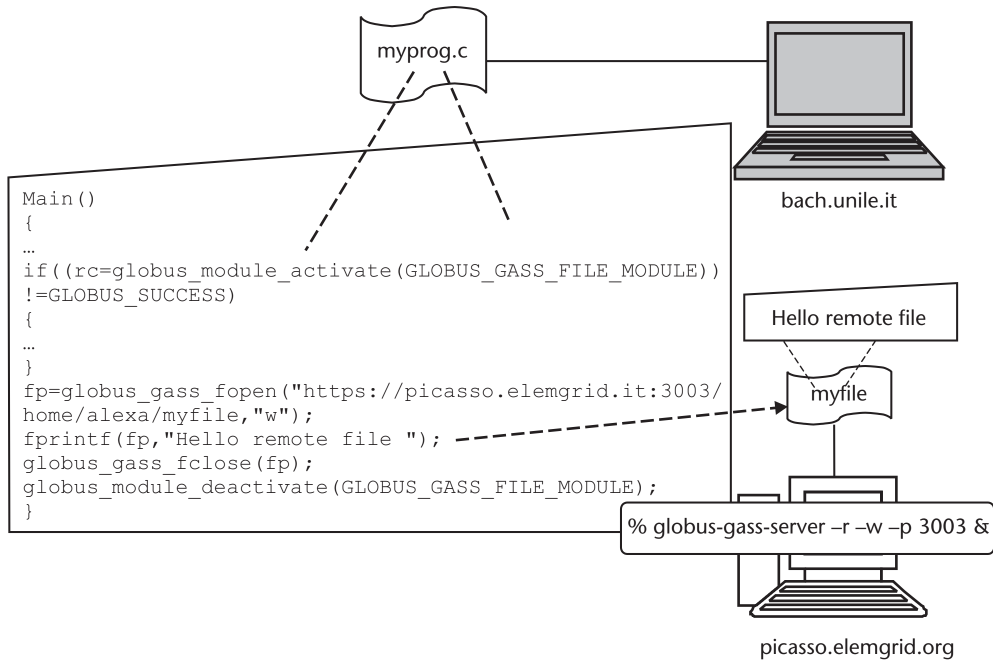

Figure 3.11 Thanks to the Globus file access API, a program can be developed that reads and writes on remote files. The program "myprog.c" is resident on the machine "bach.unile.it" and writes the string "Hello remote file" on the file "myfile" located in the machine "picasso.elemgrid.org".

```
    if((rc=globus_module_activate(GLOBUS_GASS_FILE_MODULE))
          !=GLOBUS_SUCCESS)
{
    printf("gass activation failed\n"); exit(-1);
}

    fp=globus_gass_fopen("https://picasso.elemgrid.it:3003/
          home/alexa/myfile,"w");
    fprintf(fp,"Hello remote file ");

    globus_gass_fclose(fp);
    globus_module_deactivate(GLOBUS_GASS_FILE_MODULE);
}
```

Note that the name of the file has been replaced by:

https://picasso.elemgrid.it:3003/home/alexa/myfile

which contains the protocol (HTTPS), the FQDN of the remote machine ("picasso.elemgrid.it"), the port of the remote GASS server (3003), the path to the remote file ("/home/alexa"), and its name ("myfile").

In your makefile, perform the following actions:

1. Insert the library location with the loader –L option:

-L$(GLOBUS_LOCATION)/lib

Where GLOBUS_LOCATION is the Globus installation directory ("/usr/local/globus/" in our configuration)

2. Ask to link globus GASS library with the –l option:
   -lglobus_gass_file_$(GLOBUS_FLAVOR) -lglobus_common_$(GLOBUS_FLAVOR)
   Where GLOBUS_FLAVOR is the flavor chosen during the installation of Globus ("gcc32dbg" in our installation, see also Table 3.1).
3. Express the location of the file to be included with the –I option:

   -I$(GLOBUS_LOCATION)/include/$(GLOBUS_FLAVOR)

Now you can compile your application, with your favorite C compiler. After that, to complete the procedure, on picasso, start the GASS server:

```
globus-gass-server  -r -w -p3003  &
```

and on "bach," perform the authentication step:

```
grid-proxy-init
```

The application can be launched.

3.10 Message Passing in a Grid Framework

As discussed in Chapter 1, message passing is a leading paradigm in distributed and parallel computing, with MPI the de facto standard. MPICH-G2 [5] is a grid-enabled implementation of MPI. It is implemented as one of the devices of the MPICH library, a freely available implementation of MPI.

MPICH-G2 requires the prior installation of source bundles of the Globus RM pillar. Then, the MPICH library must be configured specifying the globus2 device. The application must be compiled on each machine it is intended to run on. MPICH includes a number of compile commands (*mpicc*, *mpiCC*, *mpif90*, *mpif77*) related to the most frequently used languages. To launch an application, the command "*mpirun*" must be used. Every *mpirun* command under the globus2 device submits an RSL script to Globus. The user can follow two pathways:

1. Supply its RSL script;
2. Ask *mpirun* to construct it, based on the arguments passed to mpirun and on a file called "machines," containing the list of the machines composing the grid.

An example is now proposed, following the latter approach.

Suppose that we want now to run the application called "mympiappl" on a grid made up of two machines. First, one must write the "machines" file:

"matisse.elemgrid.org" 4

"picasso.elemgrid.org" 5

In such a case, we have indicated that machine "matisse.elemgrid.org" can host at most four processes, while machine "picasso.elemgrid.org" can host five processes in the meantime.

Now, the following sequence of operations must be repeated on each machine of the grid.

1. Create the installation directory for MPICH; for example, name it "/usr/local/mpich":

   ```
   mkdir /usr/local/mpich
   ```

2. Download MPICH (eventually in a directory other than the installation one). Extract the files from the compressed archive:

   ```
   gunzip -c mpich.tar.gz | tar xvf -
   ```

   This step creates the directory called "mpich," where the configuration scripts are located.
3. Go to the "mpich" directory created in the step 1:

   ```
   cd mpich
   ```

4. Configure MPICH specifying the "globus2" device. When configuring with the "globus2" device, you must specify one of the Globus *flavors* ("gcc32dbg" in our configuration). Also specify the chosen installation directory with the "–prefix" option.

   ```
   ./configure -device=globus2:-flavor=gcc32dbg –pre-
   fix=/usr/local/mpich
   ```

   This configuration script searches a file named "globus-makefile-header" in the directory named $GLOBUS_LOCATION/sbin. Some versions of GT put this file in the directory named $GLOBUS_LOCATION/bin. So, before launching the "configure" script, you should copy it into the sbin folder:

   ```
   cp $GLOBUS_LOCATION/bin/globus-makefile-header
   $GLOBUS_LOCATION/sbin/
   ```

5. Compile MPICH:

   ```
   make
   ```

6. Install MPICH in the directory specified by the "–prefix" option in the configuration step:

   ```
   make install
   ```

Now MPICH has been installed, and you can compile the application. Use the appropriate MPICH compile command, which depends on the language used for the application. For example, if your application has been written in the C language:

```
$MPICH_INSTALLATION_PATH/bin/mpicc –o myappl mympiappl.c
```

Where MPICH_INSTALLATION_PATH must be replaced by the directory where MPICH has been installed ("/usr/local/mpich" in our case).

Now, the application can be launched with the following command:

```
$MPICH_INSTALLATION_PATH/bin/mpirun –np
<num_processes> <name_of_application>
```

where the option "-np" specifies the number of instances of the application to be launched.

For example, consider the command:

```
/usr/local/mpich/bin/mpirun –np 7 myapp
```

In this case, seven instances of a single SPMD program named "myapp" will be created. MPICH is in charge of allocating these processes among the machines listed in the "machines" file ("picasso.elemgrid.org" and "matisse.elemgrid.org"), so that a load-balancing policy can be pursued.

3.11 Summary and Conclusions

In this chapter the main steps to building up a grid from scratch have been described, both from the system administrator and from the end user and application developer point of view. Though GT and the related software is in continuous progress, we propose to the reader a detailed guide as referred to GT version 2.2; some details could be more or less adapted to future versions. Nonetheless, the primary goal of this chapter is to guide the reader through the main practical problems to be attacked and solved when building up a grid on a Linux system, and this description and its conceptual relevance should remain valid even when different versions of the toolkit or of the Linux operating system are considered.

In summary, in this chapter, first the basic concepts on GT have been recalled. Then, the minimum requirements for hardware, software, and network are resumed to set up an effective grid. No specific requirements exist for hardware, while GT and NTP are minimum requirements for software. TCP/IP and a FQDN for each node are minimum requirements for the network. All the needed software is freeware and is both available from the Internet and from the annexed CD-ROM.

Later on, some operations on the file system and on environment variables are described, to be performed before GT installation.

Then, the installation procedure is schematized: from download, through GT installation, up to its configuration, with its security issues (authorization, authentication, and certification authority).

Service start up is the consequent step. After this, the relevant task of new user creation is described in detail, as well as a review of the most used and useful GT commands.

The development of applications, with examples related to Globus API, is then addressed, and finally how to support the message-passing paradigm in a grid is discussed.

All of the addressed subjects are approached with a practitioner-oriented method, and at the end of the chapter, the reader is assumed to be able to create a grid from scratch, first by using the software version attached in the CD-ROM, and later on with different versions of the same systems and tools.

References

[1] Foster, I., and C. Kesselman (Eds.), *The Grid: Blueprint for a New Computer Infrastructure*, San Francisco, CA: Morgan Kaufmann, 1999.

[2] Foster, I., C. Kesselman, and S. Tuecke, "The Anatomy of the Grid," *Int. Journal of Supercomputer Applications*, 2001.

[3] http://www.globus.org.

[4] Menezes, A., P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, Boca Raton, FL: CRC Press, 1996.

[5] http://www.mcs.anl.gov/mpi/mpich/download.html.

CHAPTER 4

Applications: FDTD with MPI in Grid Environments

4.1 Introduction

The FDTD method is one of the most frequently used numerical approaches in the EM community. Being a full-wave solver, it very often requires huge amounts of computational power, thus rendering the solution of large problems unaffordable with traditional workstations. Moreover, because of its algorithmic properties, the FDTD method is highly amenable for an implementation on HPC platforms. Consequently, it is an effective benchmark for a migration toward computational grids.

In this chapter, we shortly resume the method, proposing the most basic concepts and equations. Afterwards, a general discussion is proposed on the problems opened by an FDTD parallelization. A flexible parallel algorithm is proposed, and its attractive features for multiprocessor computers are discussed. The FDTD parallel implementation, based on MPI library, paves the way to a straight migration towards GC, thanks to MPICH-G2. Practical guidelines are given in order to accomplish this migration. Results are given in order to allow the reader to estimate the attainable performance, for a certain bandwidth, platform, and protocols in the GC. A benchmark on a relevant human-antenna interaction problem is proposed, for an interdepartmental computational grid. Some remarkable achievements are also focused, demonstrating that investing in GC promises wider results than the "simple" increase in CPU power or memory availability.

4.2 The FDTD Approach: Theoretical Background

4.2.1 Yee's Algorithm

The FDTD approach was introduced in a pioneering paper by Yee in 1966 [1]. Since that time, it has experienced great success, basically due to its simplicity and versatility. It was developed to solve Maxwell's curl equations in the time domain (see Appendix C), and is today one of the most used approaches for attacking partial differential equations.

Unlike several wave equation–based methods, Yee's algorithm contemporarily deals with both electric and magnetic fields, rather than solving for electric or magnetic fields alone. This allows a very easy modeling of both electric and magnetic local properties of materials and domains, exalting the flexibility of the approach.

Yee partitioned space into elementary cells. Assuming, for example, rectangular coordinates (different systems can be adopted when needed), the coordinate axes can be discretized with steps Δx, Δy, Δz. Time step is usually indicated with Δt. The generic space point P can be identified with notation ($i\Delta x$, $j\Delta y$, $k\Delta z$), or, more synthetically, (i, j, k), and any space and time function F with notation $F\big|_{i,j,k}^{n}$, this meaning that function F is computed at time $n\Delta t$, in point (i, j, k). *Yee's cell* is then obtained, as reported in Figure 4.1.

In such a cell, each H-field is surrounded by four E-field components (for instance, $H_z\Big|_{i+\frac{1}{2},j+\frac{1}{2},k+1}^{n}$ is surrounded by

$$E_x\Big|_{i+\frac{1}{2},j,k+1}^{n}, E_y\Big|_{i+1,j+\frac{1}{2},k+1}^{n}, E_x\Big|_{i+\frac{1}{2},j+1,k+1}^{n}, E_y\Big|_{i,j+\frac{1}{2},k+1}^{n}).$$

This interconnected E and H-field lattice, in conjunction with a central finite-difference discretization of Maxwell's curl equation, with second-order accuracy, leads to a set of relations with interesting properties [2]:

- The location of E and H fields in the Yee mesh implicitly enforces the Maxwell's divergence equations.
- The time-stepping process is fully explicit, and the algorithm is nondissipative: numerical wave modes propagating in the mesh do not decay because of nonphysical effects due to the time-stepping algorithm (though some discussions must be performed on numerical dispersion, as reported in Section 4.2.3).

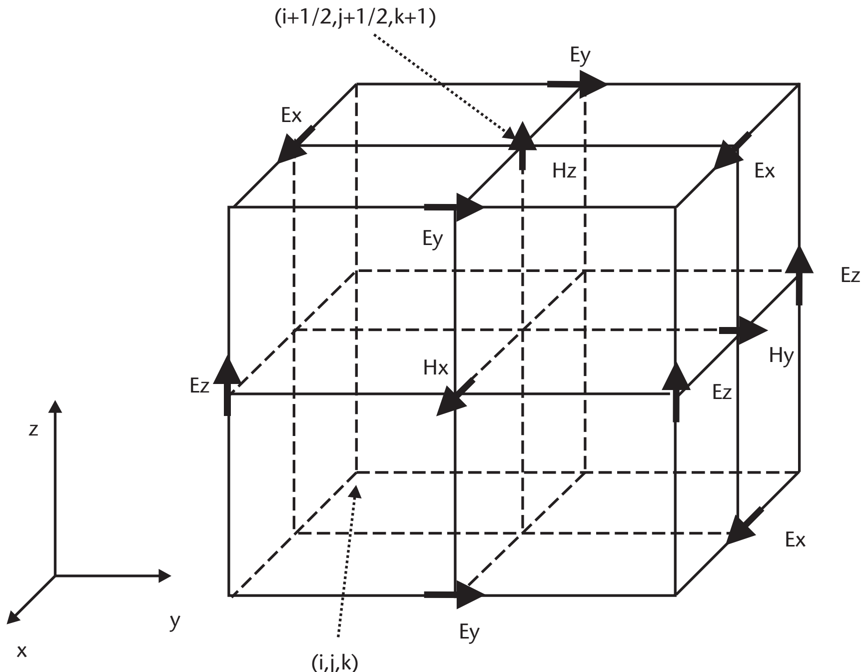

Figure 4.1 Yee's cell. Each H field component is surrounded by four E components.

The temporal and 3D spatial discretizations adopted in the FDTD algorithm are implemented at their best by using a leapfrog integration scheme to solve Maxwell's equations. In the leapfrog scheme, at time step $t = n + 1/2$, in each mesh point (i, j, k), each $\mathbf{H}^{n+1/2}$ component is computed as a function of the previous value $\mathbf{H}^{n-1/2}$ in the same point, plus a function of **E** components at time $t = n$ in the mesh points belonging to the neighborhood of (i, j, k). In a similar way, each E component is computed, at time step $t = n + 1$, in each mesh point (i, j, k), as a function of the same component at previous time step ($\mathbf{E}^n$) plus a function of the **H** components at time $t = n + 1/2$ in the mesh points belonging to the neighborhood of (i, j, k). The exact expressions for the computation are similar to the following, which is used to compute the $\boldsymbol{H}_x$ component:

$$H_x\Big|_{i,j,k}^{n+\frac{1}{2}} = D_a\Big|_{i,j,k} \cdot H_x\Big|_{i,j,k}^{n-\frac{1}{2}} = D_b\Big|_{i,j,k} * \\ * \left(\frac{E_y\Big|_{i,j,k+1/2}^{n} - E_y\Big|_{i,j,k-1/2}^{n}}{\Delta z} - \frac{E_z\Big|_{i,j+1/2,k}^{n} - E_z\Big|_{i,j-1/2,k}^{n}}{\Delta y} \right) \tag{4.1}$$

being $D_a\Big|_{i,j,k}$ and $D_b\Big|_{i,j,k}$, some constants that take into account the conductivity and permittivity of the material in each mesh point:

$$D_a\Big|_{i,j,k} = \frac{1 - \dfrac{\rho_{i,j,k}\Delta t}{2\mu_{i,j,k}}}{1 + \dfrac{\rho_{i,j,k}\Delta t}{2\mu_{i,j,k}}} \tag{4.2a}$$

$$D_b\Big|_{i,j,k} = \frac{\dfrac{\Delta t}{\mu_{i,j,k}}}{1 + \dfrac{\rho_{i,j,k}\Delta t}{2\mu_{i,j,k}}} \tag{4.2b}$$

Similar expressions can be derived for the remaining five components of E and H fields.

A quick glance at (4.1) explains why Yee's cell is conceived in such a fashion. With this artifact, combined with the leapfrog scheme, the new value for each vector component at any lattice point can be computed so that it only depends on previous values of the same component and its four adjacent components in Yee's cell. Any need for equation systems, or matrix formulations, is passed by, with a consequent relevant improvement in simplicity, efficiency, and memory requirements. The solution scheme is fully explicit; no matrix inversions must be performed.

It is also worth mentioning that the reported algorithm (differential approach) is not the unique pathway to implement an FDTD solver. An alternative, typically adopted when complex topologies must be included in the solution domain (e.g., wires or slots), is represented by the so-called integral approach, based on an integral formulation of Ampere's and Faraday's law. The elementary cell is still a

combined E-H field lattice, though in this case the major role is played by rectangular contours associated to each field component. We do not go into detail for such alternative formulation. We just recall that suitable stability conditions are needed [3] and address the reader to several papers for its implementation and application to slot modeling [4–6] or to thin wires [7, 8], just to mention some interesting contributions out of a wide variety of relevant studies.

4.2.2 Stability of the Algorithm

When a numerical approach based on a spatial mesh and a time discretization is used, a stability analysis is needed, and FDTD methods make no exceptions. The first, and still useful, studies on such a fundamental issue were performed by Taflove in 1975 [9]. The strategy was the classical Von Neumann approach, resulting in a very compact stability condition, called Courant condition. In accordance with such condition, the time step Δt must be chosen so that:

$$\Delta t \le \frac{1}{c\sqrt{\frac{1}{(\Delta x)^2}+\frac{1}{(\Delta y)^2}+\frac{1}{(\Delta z)^2}}}\,\text{sec} \tag{4.3}$$

Alternative studies have been performed (e.g., with functional analysis) leading to more general approaches. Mrozowski's contribution [10] is a reference paper for the interested reader.

4.2.3 Numerical Dispersion

As previously mentioned, a dedicated discussion must be performed on the numerical dispersion of FDTD algorithms. In fact, the phase velocity of numerical plane waves in the lattice composed of Yee's cells can depend on the wavelength, the direction of propagation, and the mesh discretization. In other words, waves propagating in the lattice can accumulate phase errors, depending on the observer's direction. This can cause relevant and nonphysical distortions in the propagation, and generally affect the accuracy of the simulation. These kinds of errors, well known and deterministic, are analyzed in detail in [2].

The error equations easily prove that numerical dispersion is theoretically more and more reduced by using a space step as small as possible. Of course, this is not compatible with actual CPU power availability. Nonetheless, starting from the dispersion equation (see Appendix C), writing it in discretized form in a 3D FDTD mesh and imposing the Courant condition (4.3) for the time step, it can be concluded that waves propagating along the mesh diagonals have no dispersions, while dispersion increases upon moving away from such directions, with a maximum for waves propagating along the axes. It is generally concluded that, when using a mesh with at least 10 cells per wavelength, the dispersion error is smaller than 1%, with values around 0.2% when 20 cells are used per wavelength. These values hold no matter what the propagation angle is and for a time step close to the limitation indicated by Courant condition. Smaller time steps can result in worse dispersion errors.

In conclusion, the choice of a space step so that at least 10 (up to 20) cells per wavelength are used, combined with a time step close to the threshold evaluated

from (4.3), is generally adopted. Anyway, this issue is still open, and several authors have proposed interesting strategies to cope with the problem of numerical dispersion, including even the use of wavelet functions in the so-called multiresolution time-domain technique [11]. A review of the dispersion of several FDTD implementations is proposed in [12].

4.2.4 Excitation and Absorbing Boundary Conditions

The excitation of sources in an FDTD code, as well as the insertion of suitable boundary conditions, is now discussed. Both themes are extremely wide, complex, and under continuous investigation. We propose a very short summarization here.

4.2.4.1 Excitations in FDTD approach

Excitations are basically partitioned into two subcases—sources outside the simulation domain and sources inside the simulation domain. The former case is quite useful when simulating the effect of a certain field impinging on the simulated region. The incident field can be modeled with the so-called total-field/scattered-field formalism: the simulation domain is divided into a total field region (inner part of the domain, including the scatterer) and an outer region (scattered field). The incident field is superimposed on the scattered field in the inner region, while it is not considered in the outer region. Details can be found in [2, 13].

As for the latter case, we put forth that the FDTD implementation we describe in this chapter deals only with this kind of excitations. Sources inside the simulation domain can be modeled with at least three different strategies—*hard sources*, *soft sources*, and *current sources*. In the hard sources approach, a source is impressed in the FDTD code by specifying the E or H field value at a specific location by a time driving function (the hard source). This is the most intuitive and trivial source excitation. The soft sources approach, on the contrary, adds the driving function to the field computed by using (4.1) and its companions (thus using the linearity of Maxwell's equations). This is physically similar to a current that is locally injected. Indeed, the application of Ampere's law to the driving function allows the derivation of an equivalent current source from the additional field term, thus leading to the current source method.

In the FDTD implementation proposed here, the hard source is used. Details on hard, soft, and current sources are available in [2].

4.2.4.2 Absorbing Boundary Conditions

A typical situation casting the problem of absorbing boundary conditions (ABC) is the case of an open region, when the simulation domain is theoretically infinite along one or more axes. The extension to infinity must be simulated somehow, as it cannot be afforded from a numerical point of view. A boundary condition is therefore needed, so that all outwards propagating waves experience no reflections when impinging on the domain's boundary. The FDTD algorithm previously described, applied to (4.1) and its companions, is not suitable to derive ABC, as it uses central differences and cannot be applied on the boundary of the mesh. A dedicated effort is therefore needed to attain suitable ABC.

ABC is probably the hottest theme in current FDTD research. Nonetheless, in accordance with the main goal of this chapter (i.e., proposing a very simple version of an FDTD code, with an MPI implementation amenable for GC), we focus now on one of the most well-known ABC, based on Enquist and Majda theory [14]: Mur's ABC. The preference for Mur's ABC is due to:

1. Their simplicity;
2. Their smaller computational demand with respect to more recent ABC, such as Berenger's ones;
3. Their amenability to parallel/distributed implementation.

Mur's ABC The starting point for Mur's ABC is the one-way wave equation [14], describing the forced propagation of a wave only along fixed directions. By using the theory of partial differential operators, as applied to Helmholtz time-domain wave equation (see Appendix C), Enquist and Majda derived two equations representing a suitable ABC. The following is the suitable case for Cartesian coordinates, for the $x = 0$ case, with a so-called first-order approximation:

$$\frac{\partial F}{\partial x} - \frac{\partial F}{c\partial t} = 0 \tag{4.4}$$

Where F is the E or H field, while the second-order approximation is:

$$\frac{\partial^2 F}{\partial x \partial t} - \frac{\partial^2 F}{c\partial t^2} + \frac{c\partial^2 F}{2\partial y^2} + \frac{c\partial^2 F}{2\partial z^2} = 0 \tag{4.5}$$

The discretization of (4.4) and (4.5) leads to Mur's first- and second-order ABC and is performed by introducing an auxiliary mesh, which is positioned one half-step inside the original mesh. The complete discrete equations are reported in the well-known paper by Mur [15].

Mur's original ABC have some limitations. Due to the existence of partial derivatives along the y and z axis, (4.5) cannot be applied on the mesh corners. Though this problem is solved by Rahhal-Arabi [16], we implement here a first-order approximation on the corners, while second-order (4.5) is considered in the remaining parts of the mesh.

It is also worth mentioning that many other authors have proposed improved versions of Mur's ABC, such as [17, 18]. Nonetheless, the current trend is oriented toward different approaches, such as perfectly matched layer (PML), by Berenger [19]. This technique imitates numerically the physical behavior of anechoic chamber's walls, where several layers of absorbing materials avoid the phenomenon of spurious reflections. Transmitted waves do not experience discontinuities in the propagation velocity, nor in the wave impedance, with respect to the impinging ones. The amount of research on PML ABC is impressive in the last years, and some papers can give an idea [20, 21].

Anyway, as previously stated, we refer here to the most basic and original Mur's ABC, appropriate for our goals of simplicity, low computational demand, and amenability to parallel migration.

4.2.5 CPU Time and Memory Requirements

The several advantages of FDTD (i.e., simplicity and versatility) are also accompanied, of course, by some drawbacks. One of the most relevant is its high computational effort, both in CPU time and memory requirements.

A very trivial analysis of the numerical complexity of a simple FDTD implementation allows us to conclude that an FDTD simulation of a domain with size $\lfloor L_x \times L_y \times L_z \rfloor m^3$ requires a number of floating point operations $NFlop = 36 \times L_x \times L_y \times L_z \times \left(\frac{N}{\lambda}\right)^3$, with N the number of samples considered for each wavelength and λ the wavelength of the EM component with highest frequency. In order to give an idea of the number of computations involved on a typical FDTD simulation, the integration of Maxwell equations on a $[100\times100\times3]\text{m}^3$ domain (a medium-sized apartment) at a frequency $f = 900$ MHz (GSM frequency) with 12 samples for wavelength and for 10^4 time steps requires nearly 14×10^{13} floating point operations. Such a large number of floating point operations clearly demonstrates that the use of FDTD methods for large EM problems compels us to adopt parallel computing techniques, with promising perspectives of significant speed ups. The exploration of supercomputing techniques and technologies is, in several cases, an effective must.

This conclusion is even strengthened when considering the issue of memory requirements. We suppose again that the spatial domain has dimensions $(L_x \times L_y \times L_z)$ m^3, and we take $N\geq10$ samples for wavelength.

Let M be the number of scalar variables to store at each mesh point. Using a material reference table with an indirect addressing scheme, seven values for each mesh point have to be kept in memory: the three electric field components E_x, E_y, E_z, the three magnetic field components H_x, H_y, H_z, and the integer value used to address the material matrix whose ith entry gives the tuple $<\varepsilon_r \mu_r \sigma>$ characterizing the ith type of material. Adopting a single precision real arithmetic (we assume this to be the best-case hypothesis, considering this sufficient for the numerical stability of the FDTD method), four bytes are required to represent a real/integer variable; 28 bytes have so to be stored at each mesh point, and the amount of memory needed to contain the whole integration domain is

$$\text{MemReq} = 28 \times L_x \times L_y \times L_z \times \left(\frac{N}{\lambda}\right)^3 \text{ Bytes}$$

Let us now consider, for instance, the simulation of the near-field zone of a radiobase antenna (RBA) for wireless communications with leading dimension D, and recall that the classical distance giving the lower limit for the far field zone [22] is $D_F = \frac{2D^2}{\lambda}$. In the case of an actual RBA with height 0.96m and working frequency 902 MHz, which is widely spread in Europe for the Global System for Mobile Communications (GSM) wireless system, the near-field zone reaches a distance $D_F = 5.76$m. In order to simulate the RBA in the whole near-field zone, the integration has to be performed on a volume of about $6 \times 6 \times 6$ m^3 which, at a discretization step set to $\lambda/18$ (for adequate accuracy), corresponds to a memory

requirement of $\text{MemReq} = 28 \times \left(6 \frac{18}{0.316}\right)^3 \approx 1.2$ GByte, thus confirming the strong need for supercomputing resources and strategies.

4.3 Parallel FDTD

The FDTD algorithm, as evinced from Section 4.2, is intrinsically amenable to deliver very high performance on parallel platforms. Its numerical characteristics allow very trivial parallelization strategies (e.g., by partitioning the simulation domain into subdomains to be independently solved by different processors or processes). This elementary form of parallelism can naturally be combined with more sophisticated and ingenious parallel strategies, thus leading to high levels of speed up.

The literature on parallel FDTD is rather rich and composite [23–26], with lots of solutions following different programming paradigms, adopting different computing platforms, and applied to different applications. This is itself a demonstration of the high flexibility and adaptability of FDTD to parallelization. Therefore, in the current section, we describe a parallel FDTD algorithm, suitable to a large variety of platforms, and demonstrate its suitability to SIMD and MIMD platforms. The same algorithm is finally migrated in Section 4.4 toward computational grids.

4.3.1 A Simple and Portable Parallel Algorithm

We propose now, in a metalanguage form, a very simple and natural parallel algorithm for FDTD. The same algorithm is then implemented with two different targets (SIMD and MIMD platforms, respectively).

On a machine with n processors, the whole computation domain is divided into n subdomains (with equal volume and shape); each subdomain is assigned to a processor and adjacent subdomains are assigned to adjacent processors. The EM field components are meanwhile updated in each processor through (4.1) and its companions (for the remaining scalar $E_{x,y,z}$ and $H_{y,z}$ components). When the computation updates a field component on the border of the domain, some values belonging to the border of the adjacent domain are required: in order to avoid communications during the computations, each subdomain is surrounded by the border cells of the other domain (as depicted in Figure 4.2 for the 2D case). These border values are communicated after the updating phase.

The scheme of the parallel algorithm is given in the following FDTD parallel algorithm:

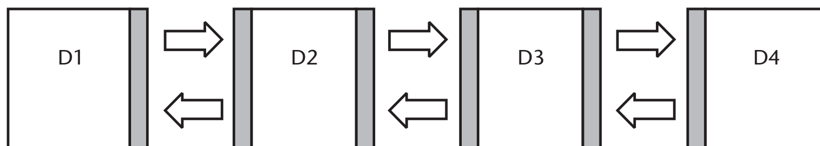

Figure 4.2 Each process is assigned data related to a subdomain, plus the borders with the adjacent subdomain, so that data communication is reduced.

Begin FDTD algorithm.

Choose a spatial discretization of the domain (Δx, Δy, Δz); if the domain has dimensions ($L_x \times L_y \times L_z$), the mesh has $N_i \times N_j \times N_k$ points, being $N_{i,j,k} = \frac{L_{x,y,z}}{\Delta_{x,y,z}} \geq 10$.

Determine the time step Δt (eq. 4.3);.

Partition the whole rectangular domain $D=[N_i \times N_j \times N_k]$ into $P= P_i \times P_j \times P_k$ rectangular subdomains $D'= [N'i \times N'j \times N'k]$, with P_i, P_j, and P_k the number of processors along dimension i, j, and k, and $N'_i = \frac{N_i}{P_i}, N'_j = \frac{N_j}{P_j}$ and $N'_k = \frac{N_k}{P_k}$ the dimension (expressed as number of mesh points) of the generic subdomain D'.

for ($t = 0$; $t < T_{end}$; $t = t + \Delta t$)

in all the processors do

compute the new values of **H**

communicate the **H** values on the boundary of each subdomain to its neighbor

compute the new values of **E**

enddo in all

put the correct value in the feed point in the processors containing the source;

in the boundary processors do compute the absorbing boundary conditions;

in all the processors communicate the **E** values on the boundary of each subdomain to its neighbor;

endfor

end FDTD algorithm.

4.3.1.1 Implementation on SIMD Platforms

An interesting example of the suitability of the FDTD algorithm for SIMD architectures is its implementation on the APE/Quadrics massively parallel system, purposely designed by Italian physicists from the National Institute for Nuclear Physics (INFN) to solve problems arising in Lattice Gauge Theory.

The APE100/Quadrics systems are characterized by 3D toroidal topology. Each node is connected to a local data memory of 4 MB (thus, the 512-node machine, the one we refer to, has 2 GB of memory).

Communications with other adjacent nodes, connected in the north, south, east, west, up, and down directions are synchronous and memory mapped; interprocessor communication bandwidth is 12.5 MBps, so the machine with 512 processor has an aggregate bandwidth of 6.4 GBbs and a peak speed of 25.6 Gflops.

The FDTD implementation on APE platform requires the use of an ad hoc programming environment (called TAO) and the related API, with a consequent lack of standardization and portability. Though this limitation is a severe drawback, the intrinsic amenability of the FDTD algorithm to accommodate itself to a SIMD philosophy allows the achievement of effective performance, as evinced from Table 4.1, where data for the efficiency E [see (1.2) in Chapter 1] are reported.

The values are referred to systems with 128 and 512 processors.

Table 4.1 Efficiency (E) on APE Platform

Domain (Cells)	*E 128 Processors*	*E 512 Processors*
262144	0,433	0,329
524288	0,459	0,365
2097152	0,586	0,477
4194304	0,614	0,519
10240000	0,694	0,598
26214400	0,762	0,668

As we can see, decreasing the domain size causes the efficiency to reduce, because the granularity of the problem becomes smaller and the overhead terms (interprocessor communications) and the less parallel parts of the code (Mur ABC) become more relevant in the global execution time.

The very attractive efficiency achieved for large problems is the clear demonstration of SIMD platforms' amenability for parallel FDTD. As a practical application, we also propose a test, concerned with the simulation of a large RBA with 12 emitting dipoles. More specifically, the antenna is produced by Kathrein and identified by the code 730691. It is shown, along with the results of the simulation, in Figure 4.3. As seen in the figure, the antenna is rather large, and it is worth noting that even larger antennas are easily encountered in European GSM network. The near-field characterization of this source requires a full-wave analysis, and this is a heavy problem for parallel FDTD. In fact, the far-field distance $D_F = \frac{2D^2}{l}$ is about 20m. Consequently, the size of the near-field sphere, as well as the required spatial and time resolution, makes the task a challenging computational effort. We simulated a domain with dimensions $(20 \times 2.5 \times 10)$ m^3 at f = 915 MHz, with N =14 samples for wavelength, for 100 periods (this simulation requires 3,330 GFlop) in nearly 18 minutes. Simulated data are in good accordance with experimental data, as shown in Figure 4.3.

4.3.1.2 FDTD Implementation with MPI

As discussed in Chapter 1, MIMD architectures are naturally open to a message-passing programming paradigm, as SPMD is the current trend in programming style. A standard for the message-passing paradigm is MPI. An MPI-based implementation of an FDTD parallel code is highly portable. The basic requirement for supporting MPI consists of a network of heterogeneous, UNIX-based platforms. Consequently, MPI appears as the most appropriate pathway to march from the very "special purpose" world of SIMD platforms to the extremely flexible direction of GC.

Indeed, as more extensively discussed in the following section, the development of an MPI implementation of the FDTD approach is the crucial step on our way to GC. Once the MPI version is ready, a very simple procedure using MPICH allows the migration to a grid.

Code Structure The proposed MPI FDTD version is based on an SPMD paradigm (i.e., a single job is composed of several instances, or *processes*, of the same

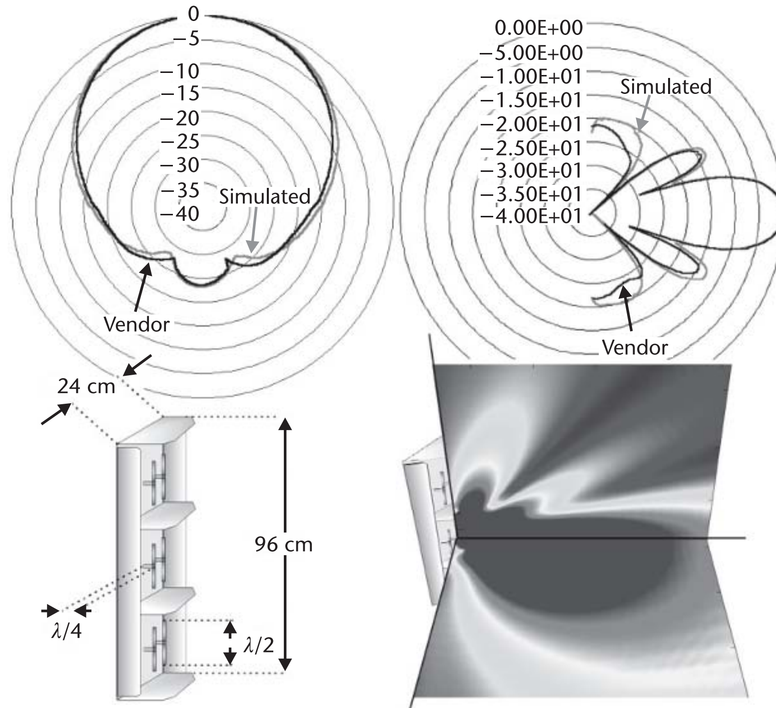

Figure 4.3 A radiobase station antenna (Kathrein 730691) used for the GSM wireless system (bottom left corner) and its radiation patters (both simulated and available from the vendor) in the E and H planes. A 3D visualization of the E-field distribution is also shown (bottom right corner).

program). The number of processes is indicated by the user when launching the application. As each processor may run several processes, suitable domain partitioning and load-balancing policies must be adopted when the number of processes is larger than the number of processors. For the sake of simplicity, we assume we deal with N_{cpu} processors and N_p processes, with $N_{cpu} = N_p = N$.

Conditional branches inside the code allow us to run different fragments of the program, depending on the process to which they belong. This is done via an identification number assigned by MPI to each process and returned by a useful function available in MPI (namely *MPI_Comm_rank*).

Processes run autonomously. This requires some synchronization to be performed. Two main pathways can be pursued to achieve the goal:

1. Use of *blocking* messages: the process sending data stops until the receiving process has completely gathered all communicated data.
2. Use of *barriers:* in MPI the function *MPI_Barrier* blocks the caller until all members of a certain group have called it (see Figure 4.4).

The former synchronization is adopted for point-to-point communications; the latter relates to collective communications. Our code recurs to barriers wherever it is necessary to synchronize the operation of processes (see the following description of *step 7*).

For further details regarding MPI, we refer the interested reader to [27].

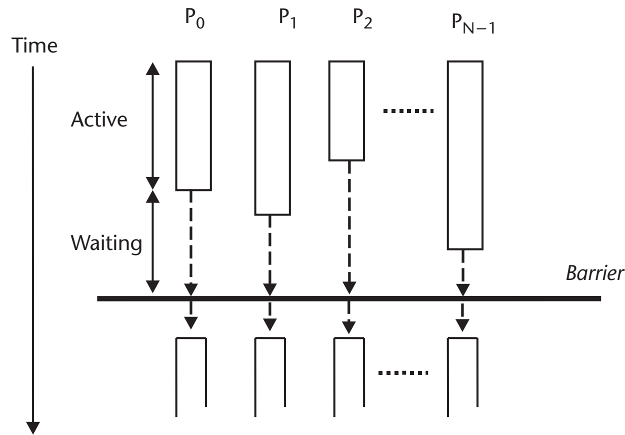

Figure 4.4 MPI barriers. Barriers synchronize process operations. When multiple processes (P_0, P_1, .., P_{N-1} in the figure) call the MPI function named "MPI_Barrier," they block until all of them have called it.

The MPI FDTD implementation (proposed in C language and reported in its most important parts in the attached CD-ROM) can be partitioned into 13 steps:

1. Definition of the simulation domain;
2. Domain partitioning into N subdomains;
3. Variable initialization;
4. Dynamic allocation of data structures;
5. MPI library initialization;
6. Evaluation of constant parameters for EM simulation and ABC calculations;
7. H-field evaluation in every cell of every subdomain;
8. Communication of H-field values, calculated along the subdomain bound, among adjacent processors;
9. E-field evaluation in every cell of every subdomain;
10. Communication of E-field values, calculated along the subdomain bound, among adjacent processors;
11. Update of excitations;
12. Mur's second-order ABC (first order on the corners) with different approaches for inner subdomains with respect to subdomains along the domain's bound;
13. Loop from step 7 up to step 12 until simulation is completed.

In Figure 4.5 a schematic diagram of the program is proposed.

Step 1 and 2 It is assumed that the end user uses his/her own preferred software tool so that the domain size and structure is defined, as well as the appropriate mesh (step 1). Once this is done, domain partitioning is performed. The whole domain is divided into N subdomains, with similar shapes and dimensions, with each subdomain associated to a processor. Though in general 3D problems and partitioning must be considered, simplified partitioning policies can be adopted,

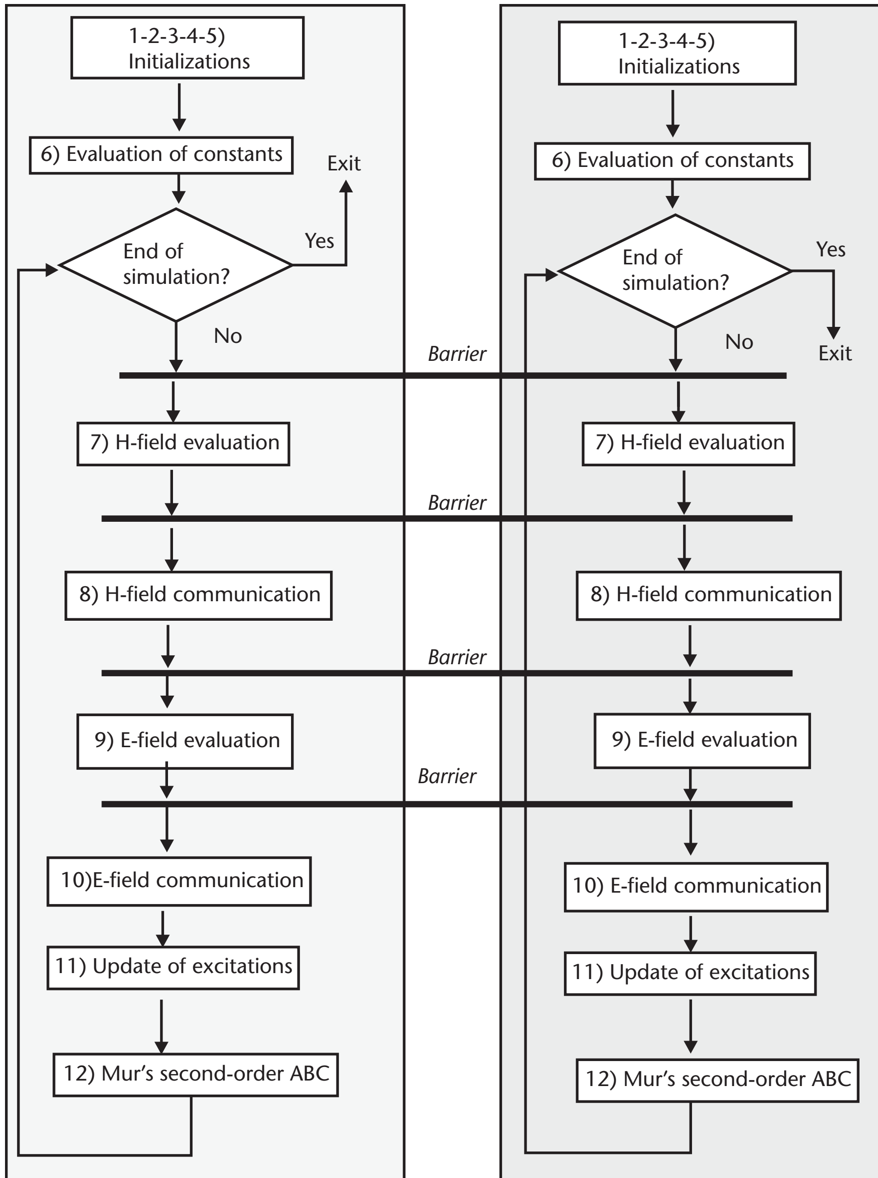

Figure 4.5 Flow diagram of the FDTD MPI code showing the steps performed by the processes. Synchronization among the processes is shown at the four barrier points. The processes cycle from step 7 to step 12 until the simulation is completed.

with a 2D or 1D Cartesian topology (see Figure 4.6), where no divisions are performed, respectively, along one or two axes.

In the proposed simple implementation, a 1D Cartesian topology is adopted (see Figure 4.7), thus reducing to two the maximum number of cell faces involved in data communication. *IM*, *JMM*, and *KMM* are the linear dimensions (in cells) of subdomains. Each domain is addressable with its own Cartesian coordinate or equivalent, thanks to the associated process identification number (see step 5).

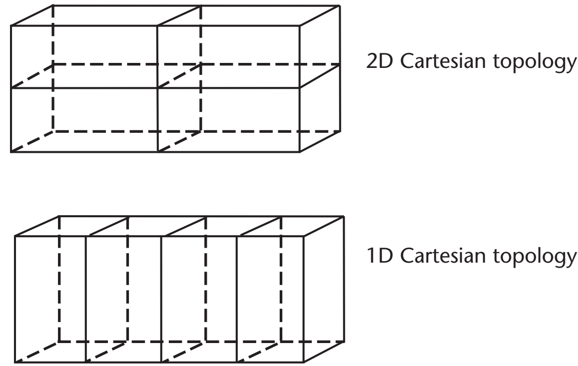

Figure 4.6 Examples of 1D and 2D Cartesian topology.

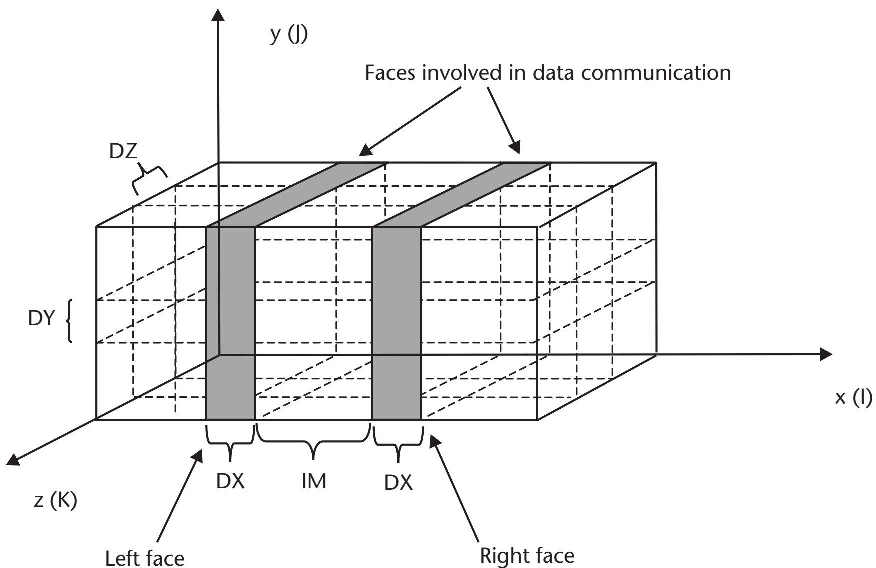

Figure 4.7 The 1D Cartesian topology, adopted to describe the code implementation. Indexes *I*, *J*, and *K* are respectively used in the code for the *x*, *y*, and *z* axes. Space steps (*DX*, *DY*, and *DZ* variables) are indicated, as well as faces involved in data communication.

Step 3 and 4 Variables must always be explicitly initialized in a suitable way, even when their initial value must be equal to zero. This is especially true in C language, as this compiler generally assigns random values to variables that are not explicitly initialized. Some relevant data structures needing such operations are the *EX*, *EY*, *EZ*, *HX*, *HY*, and *HZ* matrices. For instance, EX[I][J][K] contains the value of the x component of the *E* field in point (*I*, *J*, *K*).

They must be initialized in the following manner:

```
for (I=0;I<IM;I++)
   for (J=0;J<JM;J++)
      for (K=0;K<KMM;K++)
      {
      EX[I][J][K]=0.0;
      EY[I][J][K]=0.0;
```

```
EZ[I][J][K]=0.0;
}
```

Similar lines must be taken into account for matrices *EXYN*, *EXYN*1, *EXZN*, *EXZN*1, *EYXN*, *EYXN*1, *EYZN*, *EYZN*1, *EZXN*, *EZXN*1, *EZYN*, and *EZYN*1. These matrices contain data used when applying Mur's ABC. For instance, EXYN and EXYN1 contain the x E-field components in the two xz planes at the boundary of the simulation domain. EXYN is related to time step n, while EXYN1 is related to time step n–1. Similar considerations hold for the companion matrices.

Other relevant data structures are *C*1 and *C*2 matrices, hosting information related to the value of dielectric permittivity and conductivity in each mesh point.

It is worth mentioning that *C* compilers allocate matrix entries by rows (differently from Fortran, which order entries by columns).

It is extremely important, in order to save memory, that every memory allocation for program variables be performed in a dynamic fashion. In C language, this can be performed with the function *alloca_float* reported in the attached code, which returns a *float**** variable, and, using the malloc and memsize C instructions, dynamically allocates an A × B × C matrix:

```
float*** alloca_float(int A,int B,int C)
  { float*** a;
    int i,j,k;
    memsize=0;
    a=(float***)malloc(A*sizeof(float**));if (a==NULL) {printf("err 1\n");exit(0);}
    memsize+=A*sizeof(float**);
    for (i=0;i<A;I++)
    {
    a[i]=(float**)malloc(B*sizeof(float*));
    if (a[i]==NULL) {printf("err 2 %d\n",i);exit(0);}
    memsize+=B*sizeof(float*);
       for (j=0;j<B;j++)
       {
         a[i][j]=(float*)malloc(C*sizeof(float));
         if (a[i][j]==NULL) {printf("err 3 %d %d \n",i,j);exit(0);}
         memsize+=C*sizeof(float);
       }
    }
    return a;
  }
```

Step 5 It basically consists of three lines:

```
rc = MPI_Init(&argc,&argv);
rc = MPI_Comm_rank (MPI_COMM_WORLD,&myrank);
rc = MPI_Comm_size(MPI_COMM_WORLD,&num_tasks);
```

MPI_Init returns a return code (*rc* variable) and starts the MPI environment. MPI_Comm_rank defines the communication context (MPI_COMM_WORLD is the default communicator), returns the process identification number (*myrank*), and an error code.

MPI_Comm_size returns also the number of enrolled processes (variable *num_tasks*).

Step 6 Once the dimensions (*DX*, *DY*, and *DZ*) of the elementary cell are known, the time step DT is fixed according to the Courant condition (4.3).

Variables (*C0X*,..,*C0Z*), useful in the following part of the algorithm, as well as (A1,..., A4), used in the ABC application, are calculated once, as they do not depend on time:

```
DT=(0.999/(3E8*(pow(((1.0/DX)*(1.0/DX)+(1.0/DY)*(1.0/DY)+(1.0/DZ)*(1.0/DZ)),
0.5))));
COX=DT/(MU*DX);
COY=DT/(MU*DY);
COZ=DT/(MU*DZ);
A1=((3E8)*DT-DX)/((3E8)*DT+DX);
A2=(2*DX)/((3E8)*DT+DX);
A3=((3E8)*(3E8)*DT*DT*DX)/(2*DY*DY*((3E8)*DT+DX));
A4=((3E8)*(3E8)*DT*DT*DX)/(2*DZ*DZ*((3E8)*DT+DX));
```

Step 7 The evaluation of the H field, as required by the FDTD, is performed in the meanwhile in all processors, in every single cell of each subdomain. This (and several other issues when implementing an MPI application) casts the problem of synchronization among processes.

The code lines in charge of **H** computation are the following:

```
for (I=Iminb;I<Imax;I++)
  for (J=0;J<JM−1;J++)
    for (K=0;K<KM−1;K++)
         HX[I][J][K]=HX[I][J][K]+COZ*(EY[I][J][K+1]-EY[I][J][K])+COY*
            (EZ[I][J][K]-EZ[I][J+1][K]);

for (I=Imina;I<Imax;I++)
  for (J=1;J<JM−1;J++)
    for (K=0;K<KM−1;K++)
      HY[I][J][K]=HY[I][J][K]+COX*(EZ[I+1][J][K]-EZ[I][J][K])+COZ*
         (EX[I][J][K]-EX[I][J][K+1]);

for (I=Imina;I<Imax;I++)
  for (J=0;J<JM−1;J++)
    for (K=1;K<KM−1;K++)
      HZ[I][J][K]=HZ[I][J][K]+COY*(EX[I][J+1][K]-EX[I][J][K])+COX*
         (EY[I][J][K]-EY[I+1][J][K]);
```

These lines should be encapsulated between barriers, so that all the processors complete the **H** field evaluation at the same time.

Step 8 Data communication plays a major role in the achievement of good performance in the algorithm. As a consequence of the described structure of the FDTD algorithm, in order to update **E** and **H** fields in a generic cell (i, j, k), E and H fields in cells (i, j, k) and $(I \pm 1, j \pm 1, k \pm 1)$ are needed in the worst case. Therefore, the update of fields inside a certain subdomain does not differ from the serial case,

while data communication between adjacent processors is needed before updating cells on the boundary of subdomains. As already suggested in Section 4.3.1, the most efficient solution to reduce data communication costs is the oversizing of subdomains, as depicted in Figure 4.2. The redundancy introduced is a small memory effort but guarantees relevant speed ups.

Once H values are computed, boundary cells are communicated among adjacent processors (see Figure 4.7), so that they are ready for the evaluation of E components (step 9).

The code implementing H data communication is now proposed. In the part of code here reported, only the boundaries that face orthogonal to the x axis (whose coordinate along the x axis is identified by the *IM* variable in the code) are communicated. Indeed, these components are enough to fully calculate the EM field. It is also worth noting that, to improve efficiency, it is effective to divide processors/processes into *even* and *odd* groups. Even and odd processors perform their tasks separately (as evinced from the way MPI_send and MPI_receive are ordered in the even and odd case). This trick brings more or less improvement depending on the characteristics of the platform, with a reduction of communication times of about 20% in the most favorable cases.

```
if (myrank%2==0) /*even processor*/
{
if (myrank != (num_tasks-1))
   for (J=0; J<JMM;J++) /* HX,HY, and HZ values to be communicated are copied
onto suitable memory locations */
  {
  memcpy (&HXap[(3*J)*(KMM)], HX[IM][J],(KM)*4);
  memcpy (&HXap[(3*J+1)*(KMM)], HY[IM][J],(KM)*4);
  memcpy (&HXap[(3*J+2)*(KMM)], HZ[IM][J],(KM)*4);
   }
 if (myrank != (num_tasks-1)) /*all even processors but the last one send data to their
neighbor */
{ MPI_Send (HXap, (JM*KM*3), MPI_FLOAT, myrank+1, tag,
MPI_COMM_WORLD);}
  if (myrank!=0) /*all even processors but the first one receive from their previous
(odd) processor */
  {MPI_Recv(HYap,(JM*KM*3),MPI_FLOAT,myrank-1, tag
,MPI_COMM_WORLD,&status);}
if (myrank != 0) /*all processors that have received data copy them onto suitable
memory locations */

for (J=0; J<JMM;J++)
{
memcpy (HX[0][J], &HYap[(3*J)*(KMM)],(KMM)*4);
memcpy (HY[0][J], &HYap[(3*J+1)*(KMM)],(KMM)*4);
memcpy (HZ[0][J], &HYap[(3*J+2)*(KMM)],(KMM)*4);
}

}
else  /*for all odd processors*/
{
– Data gathering from even processors and data copy onto memory locations;
```

– Data received and relative to "right" boundary faces in even processors, are stored as "left" boundary faces;
– Data copied onto memory are sent to even processors;
}

Step 9 As noted for the H field, the high locality of data allows the contemporary updating of E fields in each processor. When using hard sources (see Section 4.2.4.1), it can be useful to impress them by enforcing a fixed time behavior of the electric field in some parts of the domain mesh (this is, just as an example, quite typical when modeling antennas or equivalent sources). In such cases, the evaluation of E fields must take into account hard sources with point, contour, or surface dedicated treatments, depending on how the excitation is expressed.

For the sake of brevity, we report now only part of the code lines for the evaluation of the E field, where the dependency of $E(i, j, k)$ on $H(i–1, j–1, k–1)$ can be noted (we recall that the very simple case of 1D Cartesian topology is addressed). Values of $H(i–1, j–1, k–1)$ are known, as they have been received from other processors in the previous step. A quick glance at (4.1) confirms that $\mathbf{E}(i, j, k)$ does not depend on $\mathbf{H}(i+1, j+1, k+1)$, thus justifying the way data have been exchanged:

```
for ( I=1; I<Imax;I++)
  for ( J=1; J<JM−1; J++)
    for ( K=1; K<KM−1; K++)
     {
      EX[I][J][K] = EX[I][J][K]*C1[I][J][K] + (HZ[I][J][K]-HZ[I][J-1][K])* \
              C2[I][J][K] + (HY[I][J][K-1]-HY[I][J][K])*C2[I][J][K];
      EY[I][J][K] = EY[I][J][K]*C1[I][J][K] + (HX[I][J][K]-HX[I][J][K-1])* \
              C2[I][J][K] + (HZ[I-1][J][K]-HZ[I][J][K])*C2[I][J][K];
      EZ[I][J][K] = EZ[I][J][K]*C1[I][J][K] + (HY[I][J][K]-HY[I-1][J][K])* \
              C2[I][J][K] + (HX[I][J-1][K]-HX[I][J][K])*C2[I][J][K];
    }
```

Step 10 Similar observations hold as for the case of step 8. In the current case, the values to be communicated are related to the "right" face of each subdomain. Therefore, each processor sends to the preceding one data related to E fields evaluated over its "left" boundary face. Even in this case, a suitable synchronization between even and odd processors is fundamental to achieve good performance.

Steps 11 and 12 As already mentioned, step 11 (excitation updating) can be efficiently implemented in conjunction with the E-field (or H-field, depending on the source to be simulated) evaluation. This is why we assume that this step, though conceptually separated from E-field evaluation, has already been addressed.

Step 12 (ABC implementation) deserves a more careful description. Mur's ABC implementation requires the memorization of E-field values along the boundary in the two time steps preceding the current step, on the two faces nearest to the boundary. Thus, code implementing ABC consists of a former step of field evaluation over the boundary, and a latter phase of field updating for the successive evaluation. Even when performing ABC, processors work independently with one another and mutually interact to communicate data relative to the cell's boundaries. For the sake of simplicity, we depict the domain partition and data communication for ABC in the

simple case of four processors and 1D topology (see Figure 4.8). Outer processors (P1 and P4) must perform ABC even on *yz* faces, while P2 and P3 must deal only with *xy* and *xz* faces. Some communications between adjacent subdomains are also needed, as shown in the picture.

We now report the code executed in a parallel fashion by all intermediate processors (P2 and P3 in the picture). This part of the code evaluates the E_z component over *xz* faces in each subdomain. In the former part of the code, Mur's first-order conditions are applied on the edges and corners, while the latter part is relative to inner values (second-order conditions).

```
for (I=1; I<IM−1; I++)
  {
    EZ[I][0][0] = EZYN[I][1][0] + A1*(EZ[I][1][0] - EZYN[I][0][0]);
    EZ[I][JM-1][0] = EZYN[I][2][0] + A1*(EZ[I][JM-2][0] - EZYN[I][3][0]);
    EZ[I][0][ KM-2] = EZYN[I][1][ KM-2] + A1*(EZ[I][1][ KM-2] - EZYN[I][0]
      [ KM-2]);
    EZ[I][JM-1][ KM-2] = EZYN[I][2][ KM-2] + A1*(EZ[I][JM-2][ KM-2]
      - EZYN[I][3][ KM-2]);
    }
for (I=1; I<IM+1; I++)
  for (K=1; K<KM−2; K++)
  {
    EZ[I][0][K]= -EZYN1[I][1][K] + A1*(EZ[I][1][K] + EZYN1[I][0][K])\
    + A2*(EZYN[I][0][K] + EZYN[I][1][K])\
    + A3*(EZYN[I+1][0][K] - 2*EZYN[I][0][K]+\
    EZYN[I-1][0][K] + EZYN[I+1][1][K] - 2*EZYN[I][1][K]\
    + EZYN[I-1][1][K]) + A4*(EZYN[I][0][K+1] - 2*EZYN[I][0][K]+\
    EZYN[I][0][K-1] + EZYN[I][1][K+1] - 2*EZYN[I][1][K] + EZYN[I][1][K-1]);
    EZ[I][JM-1][K] = -EZYN1[I][2][K] + A1*(EZ[I][JM-2][K] + EZYN1[I][3][K])\
    + A2*(EZYN[I][3][K] + EZYN[I][2][K])\
    + A3*(EZYN[I+1][3][K] - 2*EZYN[I][3][K]+\
    EZYN[I-1][3][K] + EZYN[I+1][2][K] - 2*EZYN[I][2][K]\
    +EZYN[I-1][2][K]) + A4*(EZYN[I][3][K+1] - 2*EZYN[I][3][K]+\
```

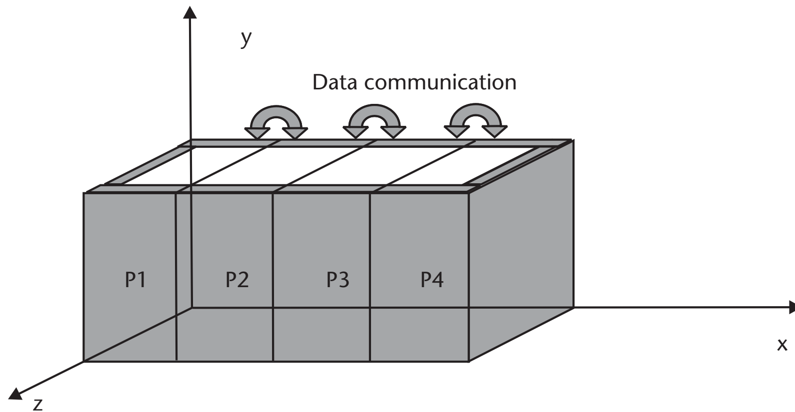

Figure 4.8 A 1D topology in the case of four subdomains. Processors P1 and P4 must deal with ABC on three faces, while P2 and P3 apply ABC on the two *xy* faces. Data communication is needed between adjacent processors (P1 with P2, P2 with P3, and P3 with P4).

```
    EZYN[I][3][K-1] + EZYN[I][2][K+1] - 2*EZYN[I][2][K] + EZYN[I][2][K-1]);
  }
```

Once the calculation of fields on outer faces is performed, the communication of data takes place. In the specific case of z E-field components, the processor receives variable $EZ(1,0, K)$ from its successive processor, and sends $EZ(IM,0, K)$ to its preceding one:

```
MPI_Send (EZ[1][0], KM+1,MPI_FLOAT, myrank-1, tag, MPI_COMM_WORLD);
    MPI_Send (EZ[1][JM-1], KM+1, MPI_FLOAT, myrank-1, tag,
MPI_COMM_WORLD);
    MPI_Send (EZ[IM][0], KM+1, MPI_FLOAT, myrank+1, tag,
MPI_COMM_WORLD);
    MPI_Send (EZ[IM][JM-1], KM+1, MPI_FLOAT, myrank+1, tag,
MPI_COMM_WORLD);
    MPI_Recv (EZ[IM+1][0], KM+1, MPI_FLOAT, myrank+1, tag ,
MPI_COMM_WORLD, &status);
    MPI_Recv (EZ[IM+1][JM-1], KM+1, MPI_FLOAT, myrank+1, tag ,
MPI_COMM_WORLD, &status);
    MPI_Recv (EZ[0][0], KM+1, MPI_FLOAT, myrank-1, tag ,
MPI_COMM_WORLD, &status);
    MPI_Recv (EZ[0][JM-1], KM+1, MPI_FLOAT, myrank-1, tag ,
MPI_COMM_WORLD, &status);
```

Finally, E-field components on outer faces, evaluated at the current and preceding time step, are updated to be used in the following of the algorithm:

```
for (I=1; I<IM; I++)
  for (K=0; K<KM−1; K++)
    {
    EZYN1[I][0][K] = EZYN[I][0][K];
    EZYN1[I][1][K] = EZYN[I][1][K];
    EZYN1[I][2][K] = EZYN[I][2][K];
    EZYN1[I][3][K] = EZYN[I][3][K];
    EZYN[I][0][K] = EZ[I][0][K];
    EZYN[I][1][K] = EZ[I][1][K];
    EZYN[I][2][K] = EZ[I][JM-2][K];
    EZYN[I][3][K] = EZ[I][JM-1][K];
    }
```

4.4 Migration Toward Computational Grids

4.4.1 Introduction

As seen in the previous sections, FDTD is amenable to parallel implementations, requiring simple domain-decomposition policies. We described in detail the specific FDTD algorithm we adopted in our experimentation, as well as a parallel implementation on SIMD and MIMD architectures. We discuss now how our FDTD implementation, parallelized using the MPI standard routines, can fruitfully and simply be migrated onto a grid environment. The result is an application that can exploit all of

the low-cost high-performance characteristics of distributed systems, as well as the flexibility of a dynamic load balancing in grids.

4.4.2 Practical Guidelines

Once the MPI version of the FDTD code has been implemented and tested, the migration toward a grid is straightforward. The code is not modified at all; the steps to perform are:

- Installation of the MPICH-G2 library at each node of the grid;
- Installation of the application source code at each node;
- Compilation of the application at each node.

MPICH-G2 is a public-domain grid-enabled implementation of the MPI standard. It is implemented as one of the devices (the "globus2" device) of the MPICH [28] library. It requires the prior installation of source bundles of the Globus RM pillar (see Chapter 3 for an explanation of how to install Globus bundles). Then, the MPICH library must be configured specifying the "globus2" device. The application must be compiled on each machine on which it is intended to run. MPICH includes a number of compile commands ("mpicc," "mpiCC," "mpif90," and "mpif77"), related to the most frequently used languages. The command "mpirun" launches an application. Every "mpirun" command under the "globus2" device submits an RSL script to Globus (see Chapters 2 and 3 for an introduction to RSL). Indeed, the Globus RM pillar is in charge of launching the jobs onto the machines and of allocating all of the resources required. The user can supply its RSL script or may ask "mpirun" to construct it, based on the arguments passed to "mpirun" and on a file called "machines," containing the list of the machines composing the grid.

Let us now introduce an example for further detailing the previously enumerated steps.

Suppose that we want to run the application called "fdtdmpi" on a grid made up of two machines, running the Linux operating system. First, we must create the "machines" file. If, for instance, the two machines' FQDN are "mozart.unile.it" and "picasso.elemgrid.org," the file appears as follows:

```
"mozart.unile.it" 4
"picasso.elemgrid.org" 5
```

In such a case, we have indicated that machine "mozart.unile.it" can host at most four processes, while machine "picasso.elemgrid.org" can host five processes in the meantime.

The installation procedure of MPICH is described in detail in Chapter 3 (Section 3.10), and is shorly resumed now. Download MPICH and repeat the following sequence of operations on *each* machine of the grid:

1. Extract MPICH files by issuing the following command:

```
gunzip -c mpich.tar.gz | tar xvf -
```

2. Configure MPICH by specifying the "globus2" device and the Globus flavor chosen during the Globus RM installation phase (see Chapter 3 for the definition of Globus flavors):

```
./configure –device=globus:-flavor=gcc32dbg
```

3. Use the makefile included with the MPICH software to compile MPICH:

```
make
```

Now that MPICH-G2 is ready, the executable "fdtdmpi" must be copied onto every node in the grid and there recompiled.

Finally, the application can be launched with the following command:

```
mpirun –np 7 fdtdmpi
```

The option "-np 7" specifies the number of instances to be launched. MPICH-G2, together with GT, is in charge of allocating these processes among the machines listed in this file, so that a load-balancing policy can be pursued.

4.4.3 Pthread Libraries and MPICH-G2

These days, the diffusion of multiprocessor architectures is more and more extended. Even entry-level platforms, such as home computers, are quite often equipped with more than one processor. This implies a high probability that the grid contains similar nodes in some of the possible dynamic configurations it might assume. When the grid contains multiprocessor nodes, the concurrency at single nodes can be exploited by demanding the computation to multiple threads. As explained in Chapter 1, programming threads is like calling procedures that run independently on the calling program. In the C language, this means that procedures are called via a specific function, normally adhering to some API specifications. To write portable applications, it is preferable to use APIs conforming to the standard specifications defined by the Institute of Electrical and Electronic Engineers (IEEE) committee. The IEEE committee defined in 1995 standard specifications for threads programming interfaces, included in the Portable Operating System for Computing Environments (POSIX) family of standards. Implementations compliant with this standard are referred to as POSIX threads, or *Pthreads* [29]. Most hardware vendors and operating systems offer their implementation of Pthreads.

Pthreads are a set of C language programming types and procedure calls, implemented with a "pthread.h" header/include file and a thread library—which may be part of another library, such as libc. The subroutines that comprise the Pthreads API can be informally grouped into two major classes: *thread management* and *synchronization*. Thread management works directly on threads—it includes functions for creating or terminating threads. Synchronization is needed to protect programs from errors due to concurrent access to shared data. In fact, threads share critical resources, such as files and memory data, with the calling program and with any other threads created by it. Consequently, some functions are available to prevent the risk of conflicting accesses to the same resource.

Functions implementing the appropriate synchronization mechanisms when accessing data are said to be *thread safe*. It means that they can be safely called by multiple threads: data are not corrupted when these functions are concurrently invoked.

MPICH-G2 is not thread safe. This means that "MPICH-G2 applications may create multiple threads but still have the restriction that *at most one thread per process may call MPI functions*" [28]. So, when implementing such an application, the developer must design with care the points where calls to MPI functions are placed.

In conclusion, we observe that the combination of GC with threads may lead to the design of dynamic applications, able to explore the grid environment in order to tailor the spawning of threads based on the availability of resources. More specifically, in the FDTD case, threads can be easily and immediately useful. A simple, yet effective, example is the following. Consider a grid made of N nodes, including one multiprocessor node (for instance, a workstation with NP processors). The policy of domain partitioning, at a former level, divides the original problem into N subproblems. By suitably using environment variables (such as variable $ARCH in UNIX-based systems), the code can identify the type of platform it is running on. The worker process running on the multiprocessor node can therefore introduce a latter level of domain partitioning, dividing the subdomain it has been assigned into NP subsubdomains, so that each subsubdomain is solved concurrently. Such a simple strategy can induce interesting speeding effects.

4.5 Numerical Performance

Proposing data of general relevance to the performance of FDTD applications in a grid is not trivial. Indeed, grids have no static topologies, and their behavior strongly depends on contingent factors, such as bandwidth availability and failures and computing loads. On the other hand, it makes sense to provide the possibility of evaluating what performance can be expected for a given grid and a certain FDTD application.

The aspects to be addressed are two:

1. How FDTD performs in a distributed environment;
2. How the MPICH-G2 framework affects performance (essentially with respect to times for message delivery).

In the following subsections, the two items are discussed, and a viable strategy is given to benchmark the performance of message passing inside the grid. Finally, data are reported related to a real grid for a specific problem.

4.5.1 Performance Evaluation of Parallel Distributed FDTD

As shown in Chapter 2, the performance of a parallel algorithm is commonly evaluated by calculating the *speed-up* factor, as well as the *efficiency*. The speed up is the ratio between the execution time on one processor and the execution time on N processors. In an ideal world, the speed-up factor increases linearly with N. In real cases, it saturates when increasing N. This happens due to two reasons:

- The program is only partially *parallelizable*;
- In message-passing systems, when N is too large with respect to the size of the problem, the communication time exceeds the computational time.

The former item scores a point for FDTD. The fraction of parallelizable algorithms is typically an attractive feature of FDTD applications. The latter item is more critical. In the wide literature available for distributed implementations of FDTD, it is shown that an adequate tradeoff must be found between the number of processors and the problem dimensions. This is especially apparent when communication times are not negligible, such as when using a network of computers rather than specialized MPPs (this being frequently the case of GC as well). Just to mention one case among the many reported in the literature, in [30] the performance of a parallel, message-passing implementation of FDTD on a supercomputer (ALEX AVX2) is compared with a network of PCs connected by a 10-Mbps Ethernet. When using the supercomputer, a speed up of 5.17 is obtained with seven processors. When using the network of PCs, the results are absolutely not satisfying because of the huge communication times, thus proving the criticality of an appropriate choice of the architecture, its size, and the available bandwidth.

In such cases, it can be important to exploit the characteristics of the platforms at their best. One of the possible suited examples is reported in [31], where a Beowulf cluster (i.e., a cluster of workstations running the Linux operating system) is used. The workstations are connected by a fast Ethernet with a maximum bandwidth of 100 Mbps. The cluster is made up of 17 nodes, each consisting of two processors. MPICH is used for message passing between nodes, while shared-memory processing within each node is enabled using the Pthreads library. The speed up saturates when more than eight processors are used, but multithreading increases the speed up by almost a factor of two when the problem size is sufficiently large.

More generally, it is demonstrated that when an appropriate balance is found between the number of nodes and the problem dimension, nearly all of the tested cases of EM FDTD applications report an efficiency not far from one [25].

4.5.2 MPICH-G2 Performance Evaluation

As discussed before, we consider the MPI standard for the development of the FDTD distributed application. The same code, containing MPI calls, can be supported by different MPI implementations, depending on the environment in which it is running. The most used MPI implementations are:

1. Native MPI;
2. Public-domain MPICH;
3. MPICH-G2.

In case 1, a version of the library dedicated to a specific platform is used, typically sold by a platform vendor along with the hardware. This guarantees high performance for the specific architecture. In case 2, we refer to a more general version of the library, not specifically tuned for any platform. Case 3 is encountered when

dealing with message passing inside the framework of grids (using GT), with MPICH-G2 the Globus-compliant MPICH implementation.

It is intuitively perceived that MPICH-G2 is exposed to the risk of degrading performance. Indeed, it is not specifically customized for any architecture. Furthermore, it must interface with GT, with a consequent additional computational effort. We report now some results quantifying the effective performance degradation. We also give a simple procedure to accomplish the same evaluation on a generic grid.

The official performance evaluation of MPICH-G2 (accomplished by MPICH-G2's developers) is available [32]. Benchmarking was done by running the performance tool named "mpptest" and distributed with MPICH in the directory named "examples/perftest. "This tool performs a classical *ping-pong* test (i.e., for each message size, a message is sent forwards and backwards between two nodes many times). The tool records the time spent for the message round trips. The program accepts as input the number of repeats for each message size, the range of the message size, and the size increment. For example, the command:

```
mpptest –size 50 500 10 –reps 100
```

asks to repeat 100 times (with the option "-reps"), the sequence of message round trips from 50 up to 500 bytes, with an increment of 10 bytes (with the option "-size").

For each set of experiments, the results are represented in graphs, with the message size on the x axis and the time on the y axis. Each graph reports three curves, as it compares the performances of MPICH-G2 with respect to MPICH-G (an old version of MPICH-G2) and to native MPI or MPICH, depending on the platform where the experiment was carried out. The evaluation consists of three groups of tests, each performed on a different platform. The first group was performed on an SGI Origin 2000 with native MPI implementation. The second group was performed on an IBM SP2 with native MPI implementation. The third was performed on a LAN connecting two SUN machines with the public domain MPICH library. In Figures 4.9, 4.10, and 4.11, we report the results available at the time of the writing of this book. They show that a limited delay is introduced by MPICH-G2 with respect to the IBM native MPI implementation and SGI native implementation, while for large message sizes MPICH-G2 overperforms MPICH in LAN environments.

It can be concluded that when referring to MPP platforms, the degradation induced by MPICH-G2 is negligible, while in the case of network environments, performance could even improve. While open to large and simple-access computational resources, GC does not substantially affect the performance of an FDTD application with respect to traditional distributed environments, and the available bandwidth remains the main bottleneck.

4.5.3 Benchmarking Parallel FDTD on a Grid

In order to benchmark the effectiveness of GC on parallel FDTD applications, we propose now some results related to the analysis of a problem relevant for human-antenna interaction studies. Results are attained using an interdepartmental grid.

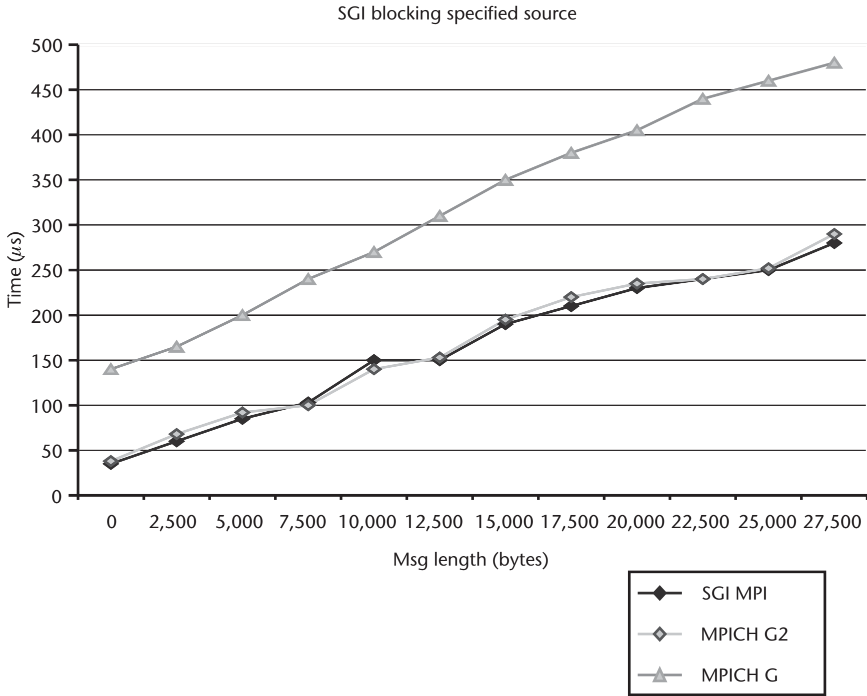

Figure 4.9 MPICH-G2 performance evaluation with respect to MPICH-G and MPI native implementation on a SGI Origin 2000 platform.

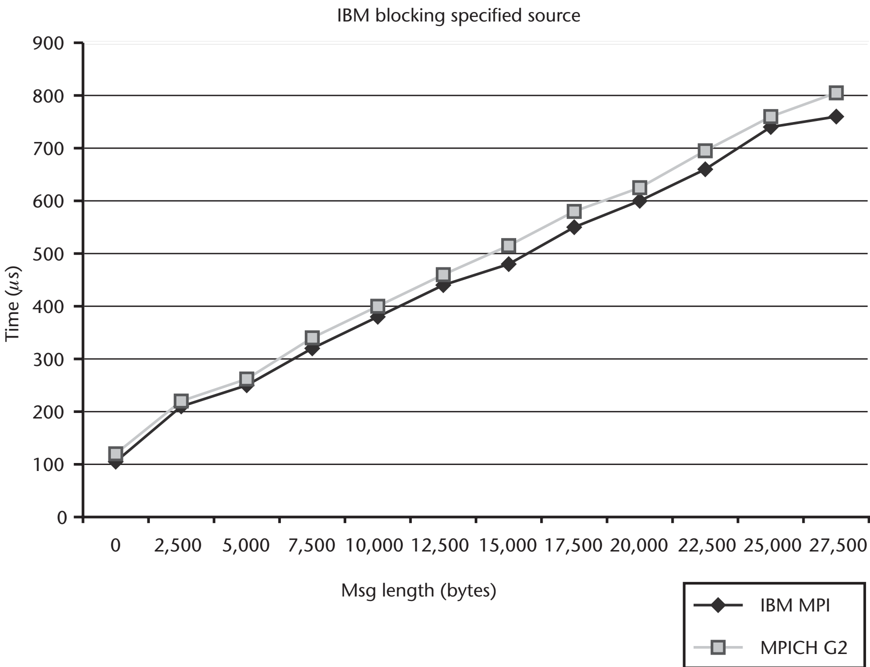

Figure 4.10 MPICH-G2 performance evaluation with respect to MPICH-G and MPI native implementation on an IBM SP2 platform.

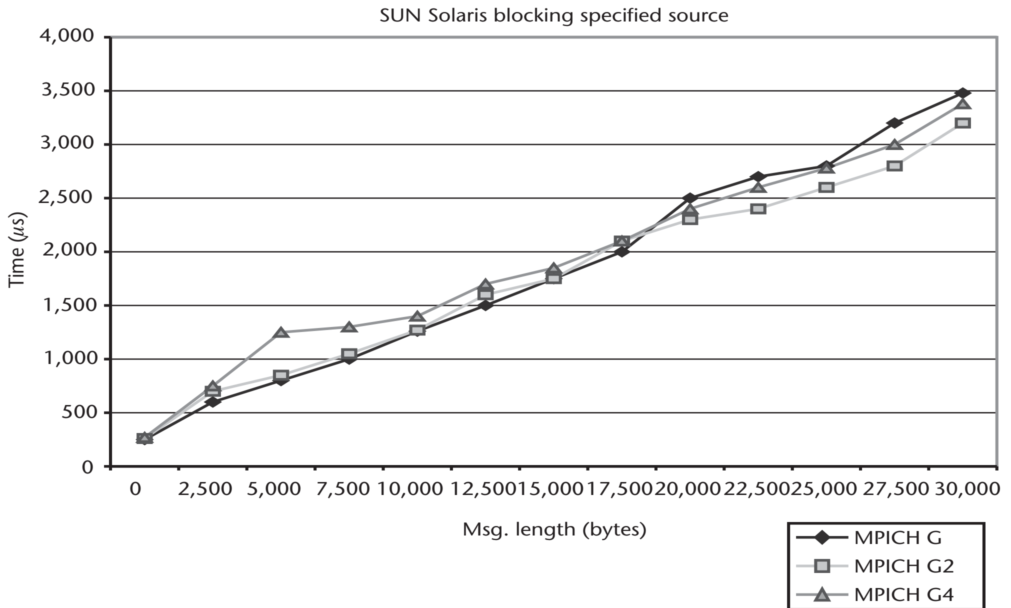

Figure 4.11 MPICH-G2 performance evaluation with respect to MPICH-G and MPICH on a LAN connecting two SUN machines.

More specifically, the grid is composed of several nodes, belonging to different departments, interconnected by Giga-Ethernet, with a low-load operating condition. The nodes in the grid selected for the benchmark are heterogeneous computers, spanning from 256-MHz up to 3-GHz processors.

The addressed problem is the analysis of a half-wave dipole radiating in the vicinity of a homogeneous dielectric sphere (see Figure 4.12), at 900 MHz. The dipole can be noted on the left of the figure, and the field distribution is evinced from gray levels.

Two cases are analyzed, with different domain sizes and the same space step, evaluated in accordance with the $\lambda/10$ principle described in Section 4.2.3. The

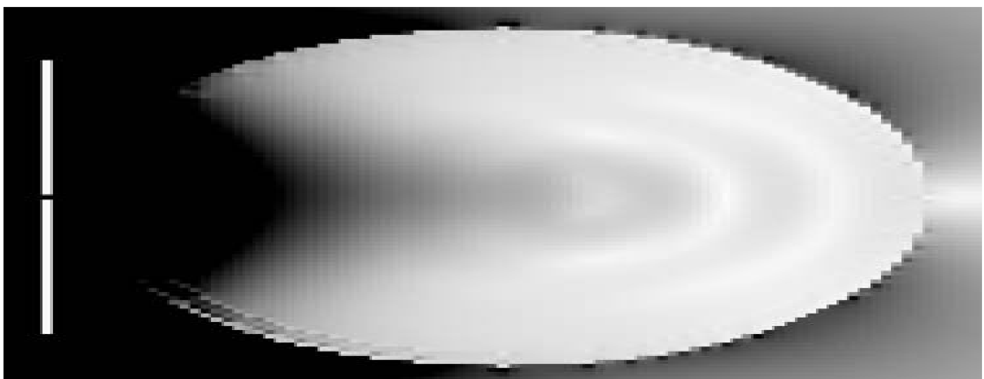

Figure 4.12 E-field levels for a homogeneous sphere exposure to the field emitted by a half-wave dipole.

former case corresponds to a cubic domain with a 256-cell edge. The latter case is a larger cubic domain with a 400-cell edge. Results are reported in Table 4.2.

As noticeable from the reported results, the speed ups look rather promising, thus confirming that GC is a cost-effective approach to improving the performance of a computationally intensive application. This is accompanied by the several advantages, peculiar to GC, described in the previous chapters and resumed in the next section.

4.6 Remarkable Achievements

On the basis of what has been observed until now, GC allows a *low-cost access* to a potentially unlimited set of *computational resources*, retaining all of the features peculiar to a parallel-distributed system. Thanks to the tools and technologies discussed in the previous sections, any applications developed using a programming paradigm based on message-passing standard MPI can be straightforwardly migrated, with no limitations with respect to the programming language used. An EM research team, with a focus on parallel FDTD methods and very basic knowledge on MPI, can afford effective parallel computing. It is not required to arrange its own supercomputing platform, nor to set up dedicated connectivity. The proposed alternative strategy is to invest in GC know-how.

Furthermore, the migration of parallel FDTD toward GC brings out as a side effect another interesting result, represented by the identification of a *complementary pathway for distributed applications* with respect to the one represented by object orientation and API for mobile agents. As seen in Chapter 2, a number of technologies are now mature for implementing distributed applications. Among them, the technology of JMA has recently demonstrated its amenability for an efficient FDTD implementation [33]. JMA requires that the application is written in Java, provided that suitable API allowing the use of mobile agents be available. Besides, each "admissible" node must host a Java virtual machine.

JMA has some drawbacks. The Java language still suffers from performance limitations with respect to the C and Fortran languages, even if it is improving continuously. Besides, the language constraint obliges users to entirely rewrite existing applications developed in languages other than Java—parallel applications cannot be migrated onto the JMA environment without a substantial rewriting in the Java language. The drawbacks put forwards now can be overcome by using GC. Indeed, GC allows users to achieve the same goals of flexibility and reconfigurability of the

Table 4.2 Speed-Ups for FDTD on a Grid

Number of Nodes	*Speed-up on a 256x256x256 Case*	*Speed-up on a 400x400x400 Case*
1	1	1
2	1.3	1.6
4	2.7	2.8
8	4.1	4.6

applications without paying a severe cost of computational performance and rewriting applications.

4.7 Conclusions

In this chapter, the general characteristics of the FDTD approach have been resumed and a general algorithm for parallel FDTD proposed. The intrinsically parallel nature of the approach renders it amenable for a large variety of parallel platforms, including both SIMD and MIMD environments. This is extremely appealing when considering a heterogeneous environment, as a grid potentially is. A *meta-application* should be considered, in charge of launching the appropriate parallel implementation on each node of the grid, including the possibility of threads where useful. The ability of GC to define a dedicated policy for load balancing and task distribution, joined with the ability to manage executables distributed in the grid (as shown in Chapter 3), demonstrates that GC fulfills all of the requirements of distributed and cooperative computing.

Moreover, the low cost required for a migration of FDTD applications towards GC makes HPC much more affordable than before, provided that suitable bandwidth is available. The use of MPICH-G2 in the framework of GT, in fact, induces a small degradation of performance with respect to traditional solutions of distributed computing. This, in conjunction with the large granularity of the algorithm, is a highly attractive feature, encouraging an investment on GC know-how, rather than on very expensive computing platforms.

Furthermore, GC is proposing itself as a complementary strategy for enabling technologies, thus opening interesting perspectives for cooperative engineering.

Acknowledgments

The authors want to express their sincere and deep gratitude to Paolo Palazzari and Luca Catarinucci. Paolo and Luca gave a fundamental contribution to the progress of the FDTD project.

References

[1] Yee, K. S., "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equation in Isotropic Media," *IEEE Trans. Antennas and Prop.*, AP-14, May 1966, pp. 302–307.

[2] Taflove, A. *Computational Electrodynamics: the Finite-Difference Time-Domain Method*, Norwood, MA: Artech House, 1995.

[3] Railton, C. J., I. J. Craddock, and J. B. Schneider, "The Analysis of General 2D PEC Structures Using a Modified CPFDTD Algorithm," *IEEE Trans. Microwave Theory Techn.*, Vol. 44, No. 10, 1996, pp. 1728–1733.

[4] Gilbert, J., and R. Holland, "Implementation of the Thin Slot Formalism in the FD EMP Code," *IEEE Trans. Nuclear Sc.*, 28, 1981, pp. 4269–4274.

[5] Riley, D. J., and C. D. Turner, "Hybrid Thin-Slot Algorithm for the Analysis of Narrow Apertures in FDTD Calculations," *IEEE Antennas Prop. Symp.*, Vol. 38, No. 12, 1990, pp. 1943–1950.

[6] Taflove, A., et al., "Detailed FDTD Analysis of EM Fields Penetrating Narrow Slots and Lapped Joints in Thick Conducting Screens," *IEEE Antennas Prop. Symp.*, Vol. 36, No. 2, 1988, pp. 247–257.

[7] Holland, R., and L. Sympson, "FD Analysis EMP Coupling to Thin Struts and Wires," *IEEE Trans. EM Comp.*, Vol. 23, No. 2, 1981, pp. 88–97.

[8] Umashankar, K. R., A. Taflove, andB. Beker, "Calculation and Experimental Validation of Induced Current on Coupled Wires in an Arbitrary Shaped Cavity," *IEEE Antennas Prop. Symp.*, Vol. 35, No. 11, 1987, pp. 1248–1257.

[9] Taflove, A., and M. E. Brodwin, "Numerical Solution of Steady-State EM Scattering Problems Using the Time-Dependent Maxwell's Equations," *IEEE Microwave Theory Techn.*, Vol. 23, No. 8, 1975, pp. 623–630.

[10] Mrozowski, M., "Stability Condition for the Explicit Algorithm of the Time-Domain Analysis of Maxwell's Equations," *IEEE Microwave and Guided Wave Lett.*, Vol. 4, No. 8, 1994, pp. 279–281.

[11] Krumpholz, M., and L. Katehi, "MRTD: New Time Domain Schemes Based on Multiresolution Analysis," *IEEE Trans Microwave Theory Techn.*, Vol. 44, No. 4, 1996, pp. 555–571.

[12] Slager, K. L., et al., "Relative Accuracy of Several FDTD Methods in 2 and 3D," *IEEE Antennas Prop. Symp.*, Vol. 41, No. 12, 1993, pp. 1732–1737.

[13] Kunz, K. S., and R. J. Luebbers, *The FDTD Method for Electromagnetics*, Boca Raton, FL: CRC Press Inc., 1993.

[14] Enquist, B., and A. Majda, "ABC for the Numerical Simulation of Waves," *Math. Of Computation*, Vol. 31, 1977, pp. 629–651.

[15] Mur, G., "ABC for the FD Approximation of the Time-Domain EM Field Equations," *IEEE Trans. EM Comp.*, Vol. 23, No. 4, 1981, pp. 377–382.

[16] Rahal-Arabi, A., and R. Mittra, "An Alternative Form for the Mur 2nd-Order ABC," *Micr. Opt. Techn. Lett.*, Vol. 9, No. 6, 1995, pp. 336–338.

[17] Higdon, R. L., "ABC for Difference Approximations to the Multidimensional Wave Equation," *Mathematics of Computation*, Vol. 47, 1986, pp. 437–459.

[18] Mei, K. K., and J. Fang, "Superabsorption—A Method to Improve ABC," *IEEE Ant. Prop. Symp*, Vol. 40, No. 9, 1992, pp. 1001–1010.

[19] Berenger, J. P., "A Perfectly Matched Layer for the Absorption of EM Waves," *Journ. Comp. Phys.*, Vol. 114, 1994, pp. 185–200.

[20] Teixeira, F. L., and W. C. Chew, "PML-FDTD in Cylindrical and Spherical Coordinates," *IEEE Mirc. And Guided Wave Lett.*, Vol. 7, No. 9, 1997, pp. 285–287.

[21] Gedney, S. D., "An Anisotropic PML Absorbing Medium for the FDTD Simulation of Fields in Lossy and Dispersive Media," *Electromagnetics*, Vol. 16, No. 4, 1996, pp. 399–415.

[22] Kraus, J. D., *Antennas,* New York: McGraw Hill, 1988.

[23] Chew, K. C., and V. Fusco, "A Parallel Implementation of the FDTD Algorithm," *Int. Journ. Num. Modeling*, Vol. 8, 1995, pp. 293–299.

[24] Gedney, S. D., "FDTD Analysis of MW Circuit Devices in High Performance Vector/Parallel Computers," *IEEE Trans. Microwave Theory Techn.,* Vol. 43, No. 10, 1995, pp. 2510–2514.

[25] Guiffaut, C., and K. Mahdjoubi, "A Parallel FDTD Algorithm Using the MPI Library," *IEEE Ant. Prop. Mag.,* Vol. 43, No. 2, 2001, pp. 94–103.

[26] Catarinucci, L., P. Palazzari, and L. Tarricone, "Human Exposure to the Near-Field of Radiobase Antennas: A Full-Wave Solution Using Parallel FDTD," *IEEE Trans. Micr. Theory Techn.*, Vol. 51, No. 3, 2003, pp. 935–941.

[27] Pacheco, Peter S., *Parallel Programming with MPI*, San Francisco, CA: Morgan Kaufman, 1997.

[28] http://www.mcs.anl.gov/mpi/mpich/download.html.

[29] Butenhof, D. R., *Programming with POSIX Threads*, Reading, MA: Addison-Wesley, 1997, pp.1–12

[30] Forenc, J., and A. Skorek, "Analysis of High-Frequency Electromagnetic Wave Propagation Using Parallel MIMD Computer and Cluster System," *Proc. International Conference PARELEC 2000*, 2000, pp. 176–180.

[31] Schiavone, G., et al., "FDTD Speedups Obtained in Distributed Computing on a Linux Workstation Cluster," *Proc. IEEE Antennas and Propagation Society International Symposium,* 2000, pp. 1336–1339.

[32] Http://www3.niu.edu/mpi.

[33] Siniaris, C. G., et al., "Implementing Distributed FDTD Codes with Java Mobile Agents," *IEEE Antennas and Propagation Magazine*, Vol. 44, No. 6, December 2002, pp. 115 –119.

CHAPTER 5

CAE of Aperture-Antenna Arrays

5.1 Introduction

As thoroughly discussed in the previous chapters, computational grids have a number of attractive features: they support low-cost, scalable, and flexible HPC environments; guarantee high security standards; offer full opening to Web applications; and boast an intrinsic alignment with new emerging software engineering methodologies. This paves the way to a wide range of possible applications of GC to the world of computational EM.

In Chapter 4, we discussed the use of GC as an innovative solution to EM problems with relevant computational weight. The FDTD simulation of human-antenna interaction problems has been considered as a reference benchmark, and the suitability of GC to support the huge computational effort has been discussed.

In this chapter, we identify another area of application, switching toward different numerical techniques (mainly in the family of *method-of-moments* approaches) and industrial processes. The addressed application is the CAE of arrays of aperture antennas.

This application represents a more difficult test with respect to the FDTD one in Chapter 4. CAE of aperture antennas, in fact, joins together severe requirements of HPC, with a strong demand for *cooperative engineering*. GC is asked to satisfy both requests, and in this chapter we explain why it represents an adequate answer.

As Chapter 4 is fully devoted to discussing the viability of HPC with GC, in the present chapter the main emphasis is on cooperative engineering and how to support it in a GC framework.

Coming to CAE of aperture antennas, the analysis and design of large antenna arrays typically belong to the class of problems of relevant complexity. The use of very large flange-mounted arrays of apertures, with even more than 100 apertures, and complicated feeding apparatuses, is routinely encountered in many telecommunication applications. It can easily happen that stringent requirements on the electrical properties of the feeding sections must be harmonized with rigid manufacturing specifications, as well as mechanical properties of the radiating apparatuses. Last, but not least, suitable yields are expected when migrating the project from a prototyping phase to production.

In such a scenario, the analysis, design, and testing steps of the array must be accompanied by suitable optimization procedures and accomplished in an integrated manner, inside the framework of a CAE tool. This is the only effective way to render times and costs suitable and guarantee a high standard of quality.

As usually happens when attacking complex processes, it is nearly compulsory to introduce some forms of schematization and simplification, so that the problem can be more easily formulated and divided into simpler subproblems. In such a perspective, the problem of CAE of devices, such as the array of rectangular flange-mounted apertures reported in Figure 5.1, can be considered as composed of four main tasks:

1. Analysis and design of the feeding waveguide section;
2. Analysis and design of the apertures over the flange;
3. Analysis of the overall behavior of the system, including mutual coupling among apertures;
4. Analysis of the consequent radiating properties of the system.

Of course, these four procedures must tightly interact and can be considered part of an iterative optimization procedure.

These four steps typically deserve different methodologies of analysis and solution, thus leading to a first reason of complexity: the high degree of heterogeneity of numerical techniques and methodologies to be harmonized. It can easily happen that these tasks can be attacked separately, by different research groups, thus joining the heterogeneity of strategies with another potential source of complexity: the geographical distribution of skills and resources that must interact and cooperate in a concurrent fashion. Furthermore, as clarified in the chapter, we deal with numerical problems implying a strong computational effort, with a consequent demand of huge computational power. Finally, as a direct implication of the necessity to integrate distributed expertise and tools, a severe problem of security must be faced, in order to guarantee to all of the subjects involved in the cooperation the ability to share their operative capacities while retaining all of the intellectual and material properties in which they are interested. This can even cast the demand for a sort of brokering entity, able also to deal with possible economical transactions among the subjects involved in the cooperation.

In other words, as anticipated before, CAE applications need HPC and would benefit by effective cooperative engineering tools. GC is an appropriate approach to fulfill both of these requirements, and this chapter explains why and how. In order to do this, it is structured as follows.

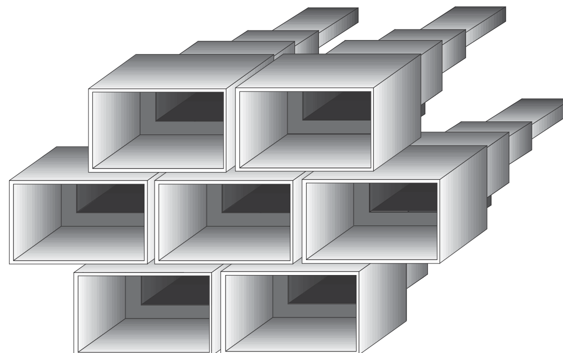

Figure 5.1 A typical array of rectangular apertures. The metallic flange is not shown, for graphical reasons. The feeding apparatus is composed of stepped rectangular waveguides.

First, Section 5.2 is focused on giving an introduction to what can be considered the theoretical core of the EM problem (i.e., the study of radiation from apertures, starting from the single-aperture case and arriving at the mutual coupling among multiple apertures). The case of infinite metal flange and rectangular apertures is considered. Later on, in Section 5.3, an integrated framework for CAE of aperture arrays is described, as composed of several (potentially independent) modules. In Section 5.4, the environment for CAE of arrays is migrated towards parallel MIMD platforms. A detailed preliminary analysis is proposed, needed to design the parallel algorithm. Later on, sophisticated parallel strategies are discussed, based on Petri nets and load-balancing policies. Some results demonstrate the attractive performance attained with an MPICH implementation. On such bases, Section 5.5 proposes the migration of the environment towards GC, describing the practical aspects and discussing the relevant implications. The reader is suggested to recall the previously enumerated issues of *heterogeneity, geographical distribution, huge computational power, security, and brokering*, so that a critical evaluation of the suitability of GC to such goals can be formulated when concluding the chapter. In other words, rather than proposing a guide to CAE of aperture-antenna arrays, we would like to give an idea of GC amenability to manage with complex and integrated environments, more and more frequently encountered in computational EM. CAE tools are a very well-suited example.

5.2 Numerical Techniques for the Analysis of Flange-Mounted Rectangular Apertures

5.2.1 Theoretical Background

Rectangular aperture antennas are routinely used for several applications, and the analysis of the basic radiating system has received considerable attention so far. Typically, for computational purposes, the presence of an infinite metallic flange has been considered because this hypothesis permits the use of free-space Green's functions, thus considerably alleviating the numerical effort. Moreover, the hypothesis is quite adherent to physical reality: in the majority of cases, the metallic flange's size fully justifies such an approximation.

A wide research effort has been produced on the subject. After the first pioneering works of [1–5], where only the contribution of the dominant mode in the aperture field was considered, several other works have been published, taking into account higher order modes or cross polarization [6–8]. A considerable variety of different numerical techniques has also been experienced (from integral equation to transverse operator techniques [9, 10]) and the relevance of the choice of different field expansions on the aperture has been studied [11].

The basic theory used to model the rectangular waveguide aperture, as well as aperture arrays, is illustrated in [12–15]. The field inside a rectangular waveguide is expressed as the sum of the modes of the waveguide (though, as extensively discussed later, this issue is open to alternative choices). Referring to transverse electric (TE) and transverse magnetic (TM) modes reported in Appendix C, and by imposing a correspondence between the variable p (or q) and the couples of indexes (m, n) typically adopted to identify modes, we can write the following:

$$\begin{bmatrix} \vec{E} \\ \vec{H} \end{bmatrix}_t^{(i)} (x, y, z) = \sum_{p(i)=1}^{M(i)} \left[a_p^i e^{-j\gamma_p z} \pm b_p^i e^{j\gamma_p z} \right] \begin{bmatrix} \bar{e} \\ \bar{h} \end{bmatrix}_p (x, y) Y_p^{\pm\frac{1}{2}} \tag{5.1}$$

In this formula, the **E** and **H** transversal components over the ith aperture are expressed as the summation of $M(i)$ modes, with **e** and **h** representing their dependence on transversal coordinates (x, y). In the same formula, if we indicate with k the free space wave number, the characteristic admittance of the mode is $Y_p = Y_o \gamma_p / k$ or $Y_p = Y_o k / \gamma_p$, respectively, for the pth TE or TM mode, and γ_p is the propagation constant for the pth mode.

Once the field has been decomposed into TE and TM modes, the elements of the aperture's generalized admittance matrix, representing the interaction between p and q modes over i and j apertures, respectively, are the following [12–14]:

$$y_{i,j}(p|q) = \frac{jkY_o}{2\pi\sqrt{Y_p Y_q}} \int\int_{D_i} dS \Psi_p(x, y) \int\int_{D_j} dS' \Psi_q(x', y') G(x - x', y - y') \tag{5.2}$$

where $G(x - x', y - y') = e^{-jk|r - r'|} / |r - r'|$ is the free space Green's function, $\Psi_p = \vec{h}_p + (\gamma_p h_{zp} / k) \vec{z}_o$—see (C.17–C.20) in Appendix C—and D_i and D_j are the ith and jth apertures (see Figure 5.2).

Equation (5.2), which holds for a generic geometry of apertures, specializes to the case of a rectangular aperture as follows:

$$y_{i,j}(m, n | m', n') = \frac{jkY_o}{2\pi\sqrt{Y_{m,n} Y_{m',n'}}} N_{C_{m,n}} N_{C_{m',n'}} \left(c_x I_x + c_y I_y - c_z I_z \right) \tag{5.3}$$

In the previous equation, c_i (with $I = x, y, z$) are evaluated as reported in [12–15], while the main computational effort is generally required by the evaluation of integrals I_i (with $I = x, y, z$), which assume the following form:

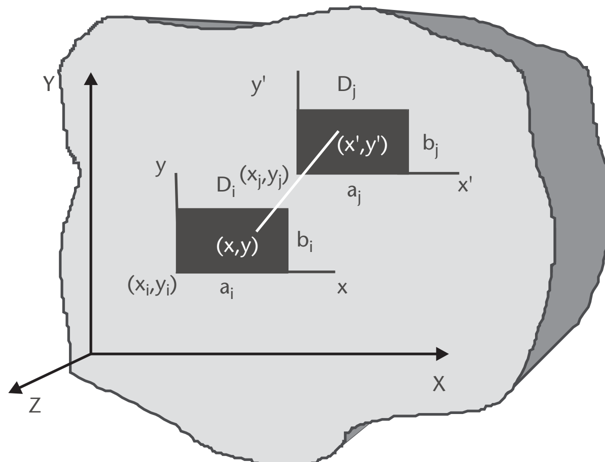

Figure 5.2 The coordinate system for a multiaperture case. D_i and D_j are the ith and jth apertures (with dimensions $a_i \times b_i$ and $a_j \times b_j$), respectively, with a relative coordinate system xy and x′y′.

$$\begin{matrix} I_x \\ I_y \\ I_z \end{matrix} = \int_{D_i}\int dS \int_{D_j}\int dS' G(x-x', y-y') \begin{matrix} \sin \\ \cos \\ \cos \end{matrix}\left(\frac{m\pi x}{a_i}\right) \begin{matrix} \cos \\ \sin \\ \cos \end{matrix}\left(\frac{n\pi y}{b_i}\right) \begin{matrix} \sin \\ \cos \\ \cos \end{matrix}\left(\frac{m'\pi x'}{a_i'}\right) \begin{matrix} \cos \\ \sin \\ \cos \end{matrix}\left(\frac{n'\pi y'}{b_i'}\right) \quad (5.4)$$

The integrations reported in (5.4) are difficult, both when a theoretical analytical solution is pursued and when approaching them with numerical techniques. It is quite apparent that a four-fold integration is not friendly, and the first step is the use of suitable transformations to reduce the order of integration.

Two main pathways will be discussed to achieve the goal: the first is based on the use of Fourier transformations (and is therefore referred to as *spectral*); the second is based on *Lewin* transformation (discussed later). Both cases are, in turn, amenable to two kinds of formulations. The former adopts waveguide modes to expand fields over the apertures; the latter, on the contrary, expands fields by means of different basis functions (e.g., Gegenbauer's polynomials [11, 16]). As a consequence, four approaches are possible (as depicted in Figure 5.3): two *spectral* methods (the former using waveguide mode expansion, the latter using Gegenbauer expansion) and two *Lewin-transformed* methods (even in this case differing with the adopted expanding functions). The methods are very shortly described later.

The efficiency in the numerical analysis of the single aperture is fundamental, especially when large arrays must be studied. In fact, the goal of an efficient analysis of multiple aperture systems cannot be even tackled without ensuring a high performance to the analysis of the single aperture. This issue is critical to setting up effective CAE environments. Consequently, this theme is specifically addressed in Section 5.3.2, when the most promising approach is identified after a rigorous analysis.

5.2.2 Approaches Based on Waveguide Modes

The two approaches are based on a field expansion over the apertures using rectangular waveguide modes. One method uses a Fourier transformation and is

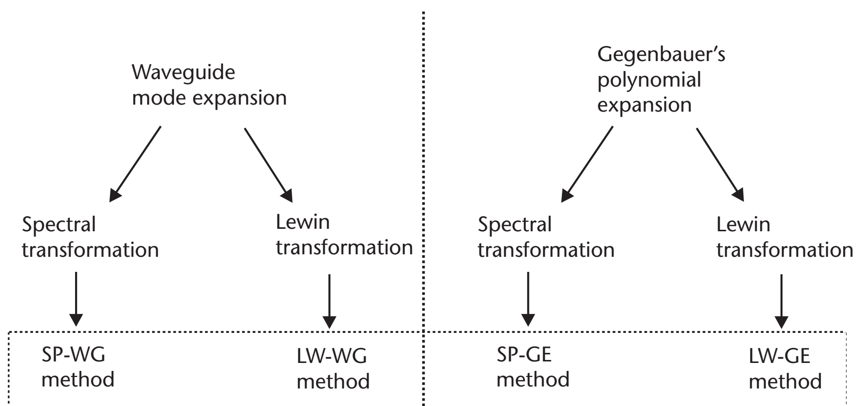

Figure 5.3 A possible schematic representation of the four formulations for the analysis of aperture arrays. The classification is performed by considering the functions adopted for the field expansion as well as the use of coordinate transformations.

addressed as spectral/waveguide (SP-WG)in the following . The other method uses a Lewin transformation, and is addressed as Lewin/waveguide (LW-WG).

5.2.2.1 Spectral Approach with Waveguide Modes (SP-WG)

The method is based on the Fourier transformation of the half-space Green's function, by using (C.23) reported in Appendix C. Thanks to suitable manipulations, (5.4) can be reduced to double integrals, as described in [17]. Such double integrations can be solved numerically, and the critical issue for an effective and accurate solution is the choice of the integration path. The use of the Fourier transformation leads to the introduction of a factor:

$$\frac{1}{\sqrt{(f_x)^2 + (f_y)^2 - \left(\frac{1}{\lambda_o}\right)^2}} \tag{5.5}$$

This renders extremely useful the following change of variables, inspired by simple trigonometric rules:

$$\begin{aligned} f_x &= f_o(\sin(\eta)\cos(\xi)) \\ f_y &= f_o(\sin(\eta)\sin(\xi)) \end{aligned} \tag{5.6}$$

The transformation (5.6) simplifies the integration, especially when an appropriate integration path is selected for the complex η plane [18]. The considered path is reported in Figure 5.4, with the real part ranging in the interval $[0; \pi/2]$, and the imaginary part in the range $[j0, j\infty]$.

This casts a rather complex problem (i.e., the identification of where to truncate the integration along the imaginary axes without affecting the accuracy of the integration). The point, related to the integration method adopted (in our case, a standard three-point Simpson method), must be empirically studied, identifying the appropriate compromise between performance and accuracy.

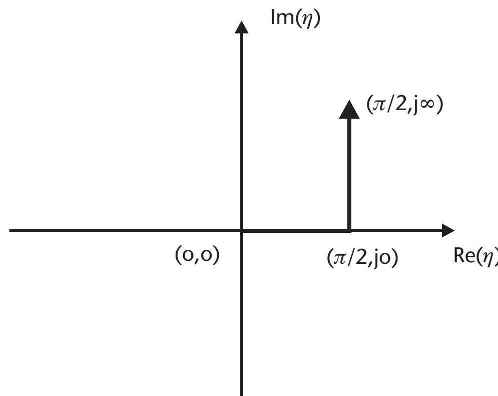

Figure 5.4 The suggested integration path in the complex η plane. A truncation is needed along the imaginary axis and represents a relevant problem to achieving a good compromise between accuracy and performance.

5.2.2.2 Lewin Approach with Waveguide Modes (LW-WG)

This method is based on the fundamental observation that (5.4) are easily reduced to the following form for the (x, x') domain (similar considerations hold for (y, y')):

$$\int_0^{a_i}\int_0^{a_j} dx' F(x-x') \begin{matrix}\cos\\ \sin\end{matrix}\left(\frac{m\pi x}{a_i}\right)\begin{matrix}\cos\\ \sin\end{matrix}\left(\frac{m'\pi x'}{a_J}\right) \tag{5.7}$$

where F is a generic function with integrable singularities. As discussed by [1, 13, 14], it is possible to reduce the order of integration by the following Lewin transformation:

$$\begin{aligned} \sigma &= x - x' \\ \lambda &= y - y' \\ \nu &= x + x' - a_i \\ \mu &= y - y' - b_i \end{aligned} \tag{5.8}$$

A double integral in the σ and λ variables is then attained, easily and efficiently solved with a Simpson rule.

5.2.3 Approaches Based on Gegenbauer's Polynomials

The two approaches are based on a field expansion over the apertures using Gegenbauer's polynomials C_m^v as expanding functions. The x and y-polarization of the E field are expanded as weighted sums of terms:

$$C_m^{1/6}(x)C_n^{7/6}(y) \quad \text{with} \quad m,n = 1,\ldots,2z+1$$

for the x polarization and:

$$C_i^{7/6}(x)C_j^{1/6}(y) \quad \text{with} \quad i,j = 0,\ldots,2z$$

for the y polarization.

This idea, proposed and discussed in [11], is based on the basic observation that these functions intrinsically satisfy the field singular behavior nearby an edge. Moreover, these orthonormal functions, though not a solution to the wave equation, generate very attractive integration kernels that are very amenable to analytical manipulation, when multiplied by the Green's half-space function. As in the case of Section 5.2.2, two methods are proposed in this section: the first uses a Fourier transform and is addressed as spectral/Gegenbauer (SP-GE) in the following. The other method uses a Lewin transformation and is addressed as Lewin/Gegenbauer (LW-GE).

5.2.3.1 Spectral Approach with Gegenbauer's Polynomials (SP-GE)

The method is based on the Fourier transformation of the half-space Green's function, as indicated in Section 5.2.2. According to [11], for the admittance matrices describing the half-space and the waveguide regions, the use of Gegenbauer's

polynomials and a coordinate transformation similar to (5.6) allow the derivation of formulas with interesting properties. The entries of the admittance matrices are complex and reduced to double integrals. The real and imaginary part of the matrix entries are attained by integrating a kernel including sinusoidal, cosinusoidal, and first-kind Bessel's functions. Such functions, coming out as a consequence of heavy and time-consuming analytical manipulations of the terms (including Gegenbauer's polynomials, which produce the attractive result of reducing the order of integration), have the drawback of introducing an oscillating behavior into the integration kernels. The real part of the admittance matrix is attained by performing the double integration over a finite domain, while the imaginary part deserves a 2D integration over an infinite domain for one out of the two dimensions. The solving formulas, allowing the evaluation of the entries of the admittance matrix (the consequent derivation of the scattering matrix is straightforward) are rather heavy and entirely reported in [17].

5.2.3.2 Lewin Approach with Gegenbauer's Polynomials (LW-GE)

The method extends to the case introduced in Section 5.2.2.2. Consequently, a Lewin-like transformation is introduced, such as the following:

$$\begin{aligned} x &= \frac{a}{2}\theta + \frac{a}{2} \\ y &= \frac{b}{2}\varphi + \frac{b}{2} \end{aligned} \tag{5.9}$$

The transformation, combined with suitable analytical developments, reduces the calculation of the coupling integrals expressing the interaction between source and test fields over the apertures to double integrals, as now reported:

$$\begin{aligned} E_{yy} &= \sum_{K=0}^{NX}\sum_{1=1}^{NY} V_k W_l \int\int F'(\theta-\theta')\left(1-\theta^2\right)^{\mu-1/2}\left(1-\theta'^2\right)^{\mu-1/2} C_i^{\mu}(\theta)C_m^{\mu}(\theta')d\theta d\theta' \\ &+ \sum_{K=0}^{NX}\sum_{1=1}^{NY} V_k W_l \int\int F''(\varphi-\varphi')\left(1-\varphi^2\right)^{\nu-1/2}\left(1-\varphi'^2\right)^{\nu-1/2} C_j^{\nu}(\varphi)C_n^{\nu}(\varphi')d\varphi d\varphi' \end{aligned} \tag{5.10}$$

where NX and NY are, respectively, the number of expanding polynomials along the x and y axes. In (5.10), the coupling between y-polarized sources and tests is evaluated. Similar equations are attained for the xx, xy, and yx cases. According to Lewin's theory, F' and F'' must fulfill the requirements of being two generic functions with integrable singularities. Though it could be assumed that these requirements are loose enough to allow the identification of two functions so that the integrations are substantially simplified, this is not the case, and the derivation of an effective LW-GE method is still an open issue.

5.3 A Tool for the CAE of Rectangular Aperture Antenna Arrays

In the previous section, we proposed a general background to the problem of the effective and efficient numerical analysis of aperture arrays. We saw that at least

four different approaches can be pursued to study the radiating properties, taking into account the relevance of mutual coupling among apertures, and we still have to identify the most suitable for our purposes.

Truly, the development of a CAE tool for such devices is a wider and more complex effort, as mutual coupling is just one of the difficulties to be faced. In this section, we propose a possible way to schematize the structure of a candidate CAE environment, by partitioning the whole task into four main subtasks and describing them separately. The following sections will focus on how to integrate them in the framework of a parallel, grid-amenable tool.

Let us assume that a system must be analyzed. It is composed of a certain number of feeders (typically horns) and the relative flange-mounted rectangular apertures. A real system was proposed in Figure 5.1, reporting a seven-aperture array for satellite applications at 3.7 GHz. The challenges of CAE of such rectangular aperture antenna arrays can be partitioned into several subproblems, namely:

1. Evaluation of the horns' scattering matrices;
2. Evaluation of the aperture array's scattering matrix;
3. Evaluation of the scattering matrix at external physical or electrical ports;
4. Evaluation of the radiation pattern.

Each subproblem can be solved by adopting different strategies and techniques, which can also be implemented by using methodologies, tools, and programming languages that can differ deeply. In other words, each subproblem can in principle be solved independently of the others by a single autonomous research group, provided that suitable communication strategies are adopted to make the different modules communicate one another (the most trivial way is communication via files). In such a way, items 1–4 can be performed sequentially, so that the radiation pattern of a given system is attained for a certain geometry, excitation, and operating conditions. The whole system, therefore, can be conceived as a single executable as well as a collection of several modules.

Some details are now given for each of the modules.

5.3.1 Evaluation of the Horns' Scattering Matrix

The analysis of the horn behavior is performed by using the mode-matching (MM) approach. Some basic guidelines for the application of the MM method are reported in the following, though we refer the interested reader to [19] for a complete description of the adopted MM formulation.

A preliminary step is the segmentation of each horn feeder into a cascade of steps in the E or H plane. This is generally needed when a tapered horn is analyzed in order to apply the basic MM assumptions. The field inside each feeder is expanded into a sum of TE and TM modes, as suggested in (5.1), and each discontinuity (step in the E or H plane) is studied by evaluating its coupling matrix **W**, whose entries are calculated with the following formula:

$$w_{ij} = \int_{S_a} E_{ia} E^{*}_{jb} dS_a \tag{5.11}$$

Where $\mathbf{E}_{ia}$ is the E field of the ith mode at a side of the discontinuity, $\mathbf{E}_{jb}$ is the E field of the jth mode at b side of the discontinuity, and S_a is the discontinuity section (see Figure 5.5). Now, letting $\mathbf{Y}_a$ and $\mathbf{Y}_b$, the diagonal matrices of the characteristic modal admittances in each waveguide (i.e., each diagonal element in $\mathbf{Y}_a$ is equivalent to the admittance of the corresponding mode in the waveguide at a side), the generalized scattering matrix for the most general case of double step can be computed according to the following formulas:

$$S = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix}$$

with

$$S_{11} = \left[Y_a + Y_c\right]^{-1}\left[Y_a - Y_c\right] \tag{5.12}$$

$$S_{21} = W\left[I + S_{11}\right] \tag{5.13}$$

$$S_{12} = 2\left[Y_a + Y_c\right]^{-1} W^T Y_b \tag{5.14}$$

$$S_{22} = WS_{12} - I \tag{5.15}$$

In (5.12–5.15), $Y_c = W^T Y_a W$. The whole waveguide structure for the horn is simulated by cascading the generalized scattering matrices of the singular step discontinuities, with the formulas reported in [20]:

$$S_{11} = S_{12a}\left(I - S_{22b}S_{22a}\right)^{-1} S_{22b}S_{21a} + S_{11a} \tag{5.16}$$

$$S_{13} = S_{12a}\left(I - S_{22b}S_{22a}\right)^{-1} S_{23b} \tag{5.17}$$

$$S_{31} = S_{32b}\left(I - S_{22a}S_{22b}\right)^{-1} S_{21a} \tag{5.18}$$

$$S_{33} = S_{32b}\left(I - S_{22a}S_{22b}\right)^{-1} S_{22a}S_{23b} + S_{33b} \tag{5.19}$$

where number 1 and 3 identify the outer physical ports at the discontinuity, while number 2 identifies the connected (inner) sections (see Figure 5.5).

5.3.2 Evaluation of the Aperture Array's Scattering Matrix

The efficiency and accuracy of the numerical approach adopted to analyze the behavior of the field over the aperture, as well as the mutual coupling among apertures, is crucial for the development of effective and viable CAD/CAE tools for antenna arrays. This compels us to make a rigorous comparison of performance and general properties of the four reviewed methods (SP-WG, SP-GE, LW-WG, and LW-GE) in order to identify the most adequate candidate to be implemented.

Due to the difficult identification of the F' and F'' functions needed to easily implement the LW-GE approach, this method is considered a future challenge and

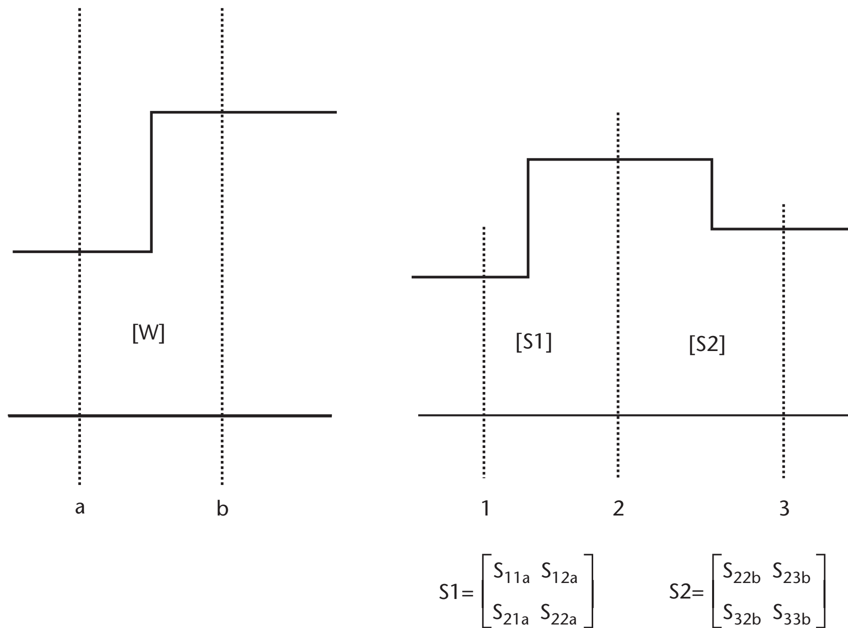

Figure 5.5 The computation of the coupling matrix **W** for a discontinuity, as referred to section a and b (left). Once the coupling matrix **W** has been determined, and the scattering matrix of the discontinuity is available, multiple discontinuities can be studied by cascading their scattering matrices. An example is reported on the right, where port 2 is a connected one, and (5.16–5.19) allow the derivation of the scattering matrix at external ports 1 and 3, provided that the scattering matrices at ports (1,2) and (2,3) have been previously computed.

not included in the present analysis. The remaining three approaches are now addressed, and some global conclusions proposed.

Out of many relevant parameters, two important factors affect the accuracy and the numerical effort required by the three approaches:

1. The number of expanding functions for an accurate description of the field;
2. The number of integration points to achieve convergence in (5.4) and its derivations.

The two parameters should be discussed jointly, and some considerations can be derived on each specific issue.

As for the dependence on the modal/polynomial set cardinality, all three methods have a quadratic complexity on the number of expanding functions. Nonetheless, this is not enough to conclude that they all have the same behavior when enlarging the number of terms in the field expansion. In fact, when keeping fixed the number of integration points, the SP-GE method is slowed less than the others by an increase in the cardinality of the modal set. This could be an advantage when many expansion terms are needed to reach convergence, such as in the case of very low aspect ratios for the apertures (squared apertures being the limit case).

As for the dependence on the number of integration points, as predictable when using Simpson integration rule, the computing time is generally proportional to the total number of integration points. Nonetheless, as in the case of the modal set cardinality, this does not mean that the three methods have the same complexity. In

this case, the LW-WG method suffers more than the others when increasing the number of points.

Finally, when performing a joint analysis, with an optimum choice of modal set cardinality and integration points so that a required accuracy is achieved, the results in Table 5.1, where the computation times are reported for the case of a squared (edge of 22.96-mm) aperture, can be reached. It is globally concluded that the SP-GE approach has a relevant drawback—the high number of integration points needed to achieve good accuracy (thus negating the advantage of a small number of expanding terms). LW-WG is generally lower performing, while the SP-WG approach is quite efficient and viable, thanks to the superior stability of its numerical kernels. The SP-WG method is then assumed to be the reference approach to be embedded in a CAE framework.

5.3.3 Evaluation of the Scattering Matrix at External Ports

Once the **S** matrix of each horn and of the aperture array have been computed (as described in Sections 5.3.1 and 5.3.2, respectively), the use of circuit theory is extremely fruitful. The situation is depicted in Figure 5.6, where an electric

Table 5.1 Computing Times for a Squared Aperture

Approach	*Comp. Time (s)*
LW-WG	1,920
SP-WG	103
SP-GE	461

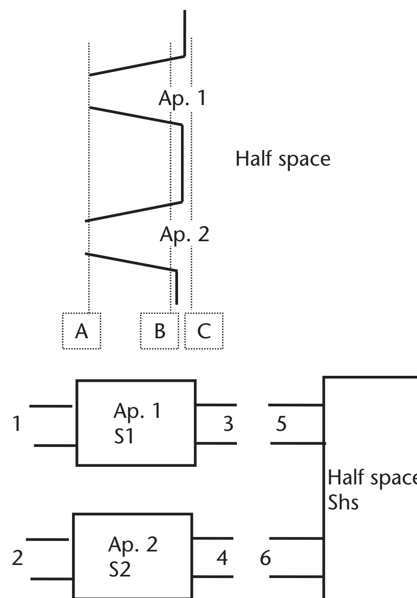

Figure 5.6 A simple two-aperture example. The scattering matrixes **S**1 for horn 1 and **S**2 for horn 2 are calculated by referring to section A and B, while the scattering matrix of the aperture array Shs is computed by referring to section C. Physical ports are numbered in the figure. For each physical port, the same number of electrical ports must be considered as the number of modes in that section.

equivalent circuit is shown for a simple case of a two-aperture array with flared horns. The two horns are represented by the scattering matrices **S1** and **S2**, evaluated with respect to section A and section B. Physical ports are numbered as reported in the figure, while electrical ports are derived in accordance with the number of modes used over physical ports to expand the fields. The aperture array's scattering matrix **Shs** is evaluated in section C. Consequently, the continuity conditions for tangential E and H components over section B and C, in terms of circuit theory, correspond to connecting physical ports 3 and 4 with, respectively, ports 5 and 6. Of course, similar considerations can be derived when referring to electrical ports.

The previous observations can be immediately translated into simple matrix algebra considerations. In fact, matrixes **S1** and **S2** generally have a blocked structure, in accordance with Figure 5.7, where the four blocks A, B, C, and D are indicated. The A and D blocks are squared submatrices, whose dimensions depend on the number of modes over ports 1, 2, 3, and 4. For instance, if we suppose that n_1 modes are used over port 1 and n_2 modes are used over port 3, matrix **S1** will have the A block with dimension $n_1 x n_1$, and D block with dimension $\mathrm{n_2xn_2}$. Accordingly, the dimension of matrix **Shs** depends on the modal expansions over the apertures adopted in section C. Now we can partition all of the physical ports into two groups: external ports (1 and 2) and internal ports (3, 4, 5, and 6). By using circuit theory and following the approach described in [20], the general scattering matrix $\mathbf{S}_G$ of the whole circuit can be derived. If we suppose that matrices **S1** and **S2** are identical, the pattern of $\mathbf{S}_G$ can be predicted and is reported in Figure 5.7. Now, four submatrices can be identified in $\mathbf{S}_G$, namely: $\mathbf{S}_{ee}$ (the scattering matrix related to external ports), $\mathbf{S}_{ii}$ (the scattering matrix related to internal ports), $\mathbf{S}_{ie}$, and $\mathbf{S}_{ei}$ (the rectangular scattering matrices related to the coupling between internal and external ports). If we suppose that the total number of modes over ports 5 and 6 is n_{hs}, $\mathbf{S}_{ee}$ is squared and has dimension $2n_1$, S_{ii} is squared and has dimension $(n_{hs}+2n_2)$, $\mathbf{S}_{ie}$ has

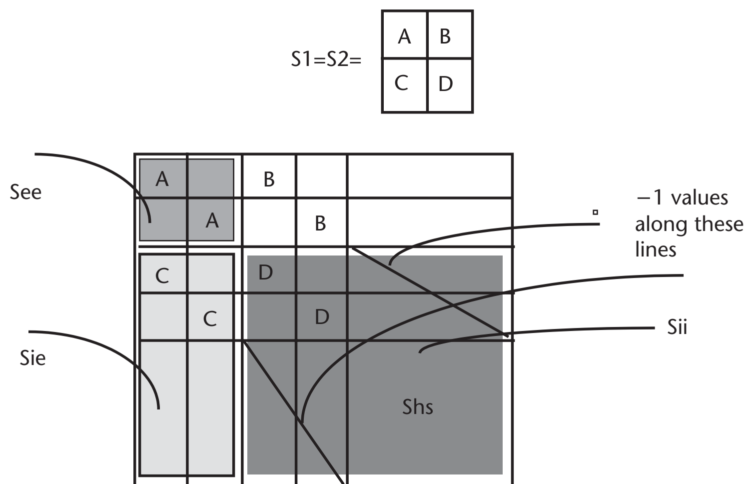

Figure 5.7 The scattering matrix of the whole device (including the feeding sections and the half space) at external ports. Once the pattern of the horn scattering matrices is known (in this case, they are supposed to be identical), the pattern of the overall S_G matrix can be predicted by using circuit theory. A connection matrix is used to attain the overall matrix, remembering the connection among internal ports.

dimension $(n_{hs}+2n_2) \times 2n_1$, and $\mathbf{S}_{ei}$ has dimension $2n_1 \times (n_{hs}+2n_2)$. The "–1" values are attained because of the interconnection among internal ports.

The same formulation based on circuit theory allows the achievement of our main goal (i.e., the derivation of the scattering matrix at external ports, which we indicate with $\mathbf{S}_e$). Indeed, if we introduce the *connection matrix* Γ, (i.e., a 0–1 symmetric matrix, whose generic entry Γ_{ij} is 1 if and only if ports *i* and *j* are connected each other), the following formula can be derived

$$S_e = S_{ee} + S_{ei}(\Gamma_i - S_{ii})^{-1} S_{ie} \tag{5.20}$$

Γ_i is the connection matrix for internal ports. The evaluation of the general matrix $\mathbf{S}_G$ is an important achievement. In addition to the evaluation of $\mathbf{S}_e$, it allows the evaluation of voltages and currents related to the modal sets used over the apertures (internal ports). In fact, if we indicate with $\mathbf{a}_i$ and $\mathbf{b}_i$ the forward and backward waves at port i, respectively, the following equations can be derived:

$$a_i = (\Gamma_i - S_{ii})^{-1} S_{ie} a_e \tag{5.21}$$

$$b_i = S_{ii} a_i \tag{5.22}$$

Consequently, voltages $\mathbf{V}_i$ can be calculated as $\mathbf{V}_i = \mathbf{a}_i + \mathbf{b}_i$, and currents as $\mathbf{I}_i = \mathbf{a}_i - \mathbf{b}_i$.

In other words, the method discussed in this section allows the derivation of a scattering matrix related to a multiport circuit whose ports are the feeding sections of the horns (external ports), as well as the consequent evaluation of voltages and currents at internal ports. This guarantees the complete characterization of the circuit, both at external and at internal ports. While the characterization at external ports is fundamental for system engineering and design, the ability to model the behavior at internal ports is a key factor when investigating the radiating properties (as discussed in the next section) and when performing optimization tasks. Indeed, the application of very efficient methods, such as the Adjoint Network Method [21, 22], is strictly correlated with such capability and represents a formidable improvement in optimization times when compared with the large majority of alternative optimization policies.

5.3.4 Evaluation of the Radiation Pattern

The evaluation of the radiation pattern is rather straightforward, thanks to the accomplishment of the operations described in the previous sections. Indeed, referring to the reference system of Figure 5.8 and adopting the classical formulation reported in [23], the electric field in a generic point is:

$$E(r) = \frac{jk_o \cos\theta}{2\pi r} e^{-jk_o r} \int\int E_a e^{j(k_o \sin\theta \cos\varphi x + k_o \sin\theta \sin\varphi y)} dxdy \tag{5.23}$$

where $\mathbf{E}_a$ is the electric field over the metallic flange. $\mathbf{E}_a$ is easily evaluated as an expansion of modes over each aperture, as the voltages $\mathbf{V}_i$ evaluated with the method

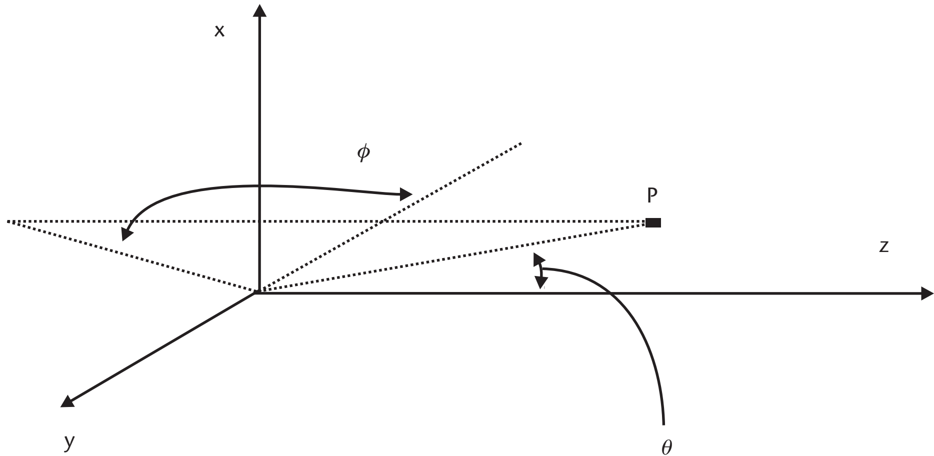

Figure 5.8 The coordinate system adopted when evaluating the radiation pattern. The third definition by Ludwig is assumed when defining the radiation pattern.

of (5.21) are exactly the weighting factors for all of the modes. Therefore, the kernel of the previous integral is composed of linear combinations of TE and TM modes of rectangular waveguides and can be considered as a Fourier transformation of cosinusoidal and sinusoidal functions.

The generic form of the integration to be performed is:

$$\int_{x_{cn}}^{x_{cn}+a_n} \sin\left(\frac{n\pi x}{a_n}\right) e^{jk_o \sin\theta \cos\varphi x}\, dx \tag{5.24}$$

where n is a modal index, a_n is the nth aperture's horizontal dimension, and x_{cn} is the nth aperture's center x coordinate, with respect to a common coordinate system.

Of course, the solution of (5.24) and the appropriate projection of E-field components along the radial direction allows, in the far-field approximation, the derivation of copolar and cross-polar radiation patterns, according to the most typical and well-known definition by Ludwig [24]:

$$\begin{aligned} E_{copol} &= E_\theta \sin\varphi + E_\varphi \cos\varphi \\ E_{cross} &= E_\theta \cos\varphi - E_\varphi \sin\varphi \end{aligned} \tag{5.25}$$

5.4 Parallel CAE of Aperture Arrays

In the previous section, we described the nature of the four main building blocks of an environment suitable for CAE of aperture arrays. From now on, we refer to the four building blocks by introducing the following acronyms:

- Evaluation of the horns' scattering matrices: analysis of the feeding system (AFS);
- Evaluation of the aperture array's scattering matrix: analysis of mutual coupling (AMC);

- Evaluation of the scattering matrix at external physical or electrical ports: evaluation of the scattering matrix (ESM);
- Evaluation of the radiation pattern (ERP).

As discussed earlier, some of the numerical tasks are extremely heavy from a computational point of view. Both the mutual coupling of apertures and the evaluation of the generalized scattering matrix of the whole circuit are complex. The former is due to integration problems; the latter is due to the implied matrix algebra, with several large matrix inversions—see (5.20). Moreover, the four building blocks are based on rather different formulations and approaches and can often be developed in the framework of large projects by different research groups with different software methodologies. Consequently, the effective implementation of an environment for CAE of aperture horn arrays must basically face two problems: the need to sustain high computing performance and the need to allow cooperative engineering. In this section we demonstrate how parallel computing can solve the first problem. In the remaining part of the chapter, it is demonstrated that GC, while supporting cost-effective parallel computing, allows a complete and secure solution to the second problem as well.

5.4.1 Preliminary Analysis

Once the most efficient formulations and numerical techniques are used for the analysis of each of the four main blocks (MBs), it is a matter of fact that the analysis of arrays with several apertures (tens or more, as often encountered in real applications) is still unaffordable with ordinary computing strategies. This is especially true when optimization strategies must be adopted, thus requiring iterative analysis.

The development of parallel solutions for method of moments/mode matching–based approaches is not trivial and must be forwarded, as is useful in the majority of cases, by a rigorous profiling of the serial version of the code in order to arrange an efficient parallel algorithm. This allows the identification of all of the possible levels of concurrency among several parts of the code, as well as of the most CPU-demanding tasks.

The first step of this preliminary analysis is the fragmentation of each of the four MBs into what we call *atomic subtasks* (ASs). For instance, the module AFS can be partitioned into seven ASs, as reported in the following Table 5.2.

Once the ASs have been identified, a time profiling is performed, and freeware software, such as *gprof* GNU profiler [25], can be used for this goal. The profiling

Table 5.2 Atomic Subtasks in AFS

Acronym for the AS	*Description*
AFS1	Data input processing
AFS2	Mode evaluation and ordering in accordance with cut-off frequencies
AFS3	Evaluation of mode numbers over apertures
AFS4	Generation of output files
AFS5	Evaluation of coupling matrixes for all of the sections
AFS6	Analysis of E/H plane steps and the relative S matrices
AFS7	Evaluation of the scattering matrix for the whole horn (for each horn)

shows that the computation of the scattering matrix that describes the interaction of modes over apertures dominates all the other subtasks (as is easily predictable), and other challenging tasks are both the computation of the **S** matrix of the single feed intended as a cascade of discontinuities (5.16–5.19) and the evaluation of the inward and outward waves at internal electrical ports (5.21–5.22).

The profiling is a fundamental step to address the parallelization effort toward the most challenging ASs and to identify data dependencies among ASs and concurrencies. Indeed, apart from more immediate levels of parallelism, such as the analysis of all of the horns, which can be performed in parallel, the profiling demonstrates that several ASs belonging to different MBs can run concurrently. Last, but not least, it generates a hierarchical tree of dependencies among ASs: each node is a subtask, and edges connecting ASs are related to data exchange among subtasks. This hierarchical tree can be fruitfully represented by adopting the schematism of *Petri nets* [26], which is a powerful way to map such dependencies when large numbers of ASs must be addressed.

This is not the appropriate context for a detailed discussion on Petri nets. However, they can quickly be described by referring to a set of *places*, a set of *transitions*, a set of *input functions*, and a set of *output functions*. Figure 5.9 shows a four-place net. Places correspond to nodes. Transitions correspond to connecting points among places (bars T1 and T2 in the figure). An input function describes the places from which a single transition can be reached, and an output function describes the places reached from a certain transition. In this figure, places are nodes 1, 2, 3, and 4; transitions are the bars T1 and T2; the input function relative to T1 is represented by nodes 2 and 4; and the output function for T1 coincides with node 1. Arrows originating from a place can be marked by a token (the black dot in the figure). When each arrow reaching a transition is marked by a token, the transition is "ready to fire." When the transition "fires," all of the arrows reaching that transition are removed from the token, while all of the nodes belonging to the output function of the transition are marked with a token. For instance, in Figure 5.9, T1 is ready to fire. When it fires, the net of Figure 5.10 is attained.

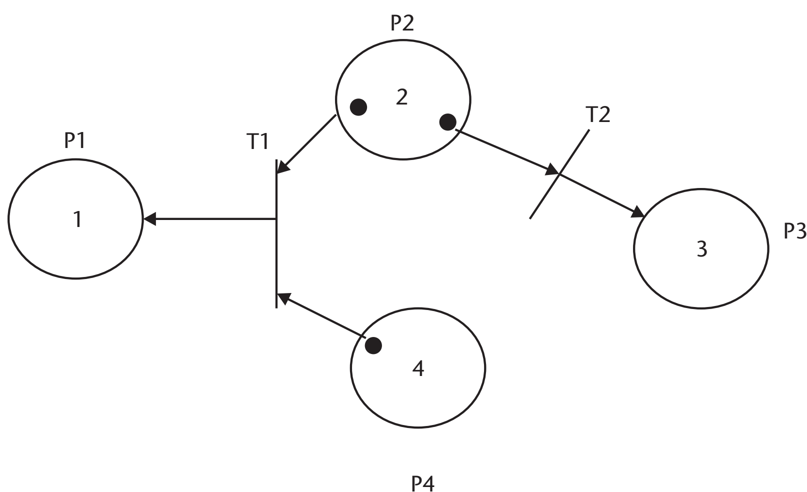

Figure 5.9 A Petri net composed of four places. Bars T1 and T2 represent transitions. Transition T1 is ready to fire, as each arrow reaching it is marked by a token.

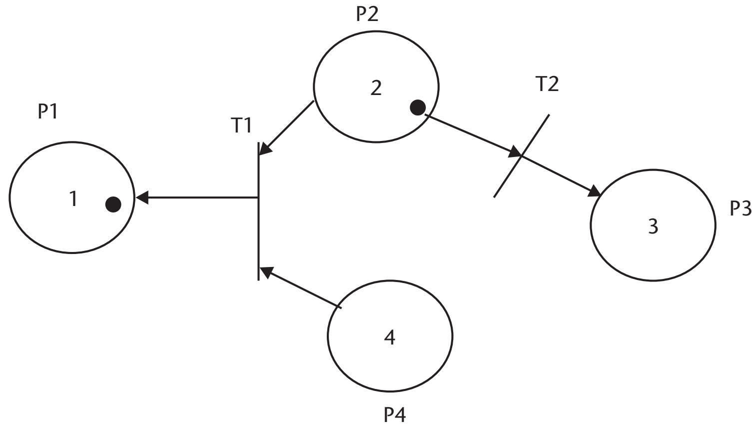

Figure 5.10 The Petri net derived by the one in Figure 5.9, after that transition T1 has fired.

Now, if we assimilate each AS to a node, data flow among ASs can be mapped by means of transitions and tokens, with transitions representing synchronization points. Petri net methodology is easily implemented in a software program, which allows the dynamical monitoring of data dependencies. The result of such an analysis is the identification of a complete net of dependencies and concurrencies among all of the ASs. This allows the identification of several levels of parallelism and has been a key factor in the design of the parallel algorithm.

The most general and simple level of parallelism is represented by the performance of the same task at different working frequencies (a typical situation can require the analysis in a bandwidth of several gigahertz, with a frequency step of tens or hundreds of megahertz). A second level of parallelism is at a geometric level—partitioning the whole circuit into subregions to be processed in parallel:

- Feeds;
- Domain of the radiation pattern (radiating half space);
- The flange and the mutual coupling among apertures.

A third level of parallelism (nested inside the former two levels) is inside heavy and crucial ASs: subtasks deserving high computational effort (especially when matrix inversion or matrix-matrix algebra is needed) can themselves be parallelized.

In several cases, the three levels can be implemented together. For instance, the AFS MB can be performed in parallel for different frequency points by partitioning frequencies through the available processors. Meanwhile, at each frequency, different sections of the feed are analyzed by different processors. When worthwhile, some specific ASs (e.g., AS AFS7 of Table 5.1, in the current example) can in turn be implemented in a parallel fashion. Anyway, it is extremely important to evaluate all of the expected benefits of a multilevel parallelism due to its nontrivial design and implementation.

The most basic evaluation to be performed is the overall numerical complexity of the aperture array analysis. The theoretical outline reported in Section 5.3 for all four MBs and the profiling described in the current section have allowed the estimation of the numerical complexity with respect to some relevant parameters, namely:

1. N_{ipr} . This is the number of integration points along the real axis in the complex η plane mentioned in the solution of (5.4) after (5.6).
2. N_{ipi}. This is the number of integration points along the imaginary axis in the complex η plane mentioned in the solution of (5.4) after (5.6).
3. N_F. This is the number of frequency points.
4. N_{map}. This is the total number of modes adopted over all the apertures.

The result of the complexity analysis is the following number F of floating point operations per second (FLOPS):

$$F = F_0 N_{ipr} N_{ipi} N_F N_{map}^2$$

where F_0 is a constant in the order of thousands. In the case of 10 apertures, with 20 modes over each aperture, the only evaluation of the mutual coupling among apertures for four frequency points directly leads to the realm of teraFLOPS! This is a convincing argument to attack the challenge of parallelization.

5.4.2 Parallelization

5.4.2.1 Load-Balancing Issues

A multilevel parallelism requires sophisticated software engineering and, above all, a suitable strategy to control load-balancing among processors and an appropriate policy for scheduling of ASs on the different processors. This is extremely important, especially in the perspective of parallelization in distributed and heterogeneous environments.

The key point in solving these problems is represented by the algorithm of recursive bisection, which is a domain-decomposition policy based on the recursive partitioning of a single domain into a couple of smaller subdomains. When the two subdomains imply identical computational effort to be analyzed, we have a balanced bisection; otherwise, it is unbalanced. The preference for balanced or unbalanced bisections depends on the platforms and their working conditions as well as on the amount of data communication implied by the domain partitioning. In fact, in several cases, the most obvious choice for a balanced policy is not advantageous, because of the induced need for communication among subdomains.

The application of the bisection algorithm is intrinsically based on the profiling action previously described. In fact, each potential domain partition is assigned a certain computational weight w_i, which can be estimated due to the Petri nets method.

The issue of optimizing the level of recursion and the scheduling policy is equivalent to the identification of an optimum assignment of one or more subdomains to the available processors, taking into account the current load and performance of each processor. Suppose that n_p processors are available, with different performance and load. The status of each processor can be described by a certain weight h_j estimated for each processor, so that h_j indicates the expectable performance, at that moment, by the jth processor. We also assume that the whole domain is partitioned into n_s subdomains $a_1, a_2, \ldots, a_{ns}$.

The problem can now be formulated for the identification of n_p groups of subdomains a_i, so that the total computational weight for each processor is well suited to its available computational power, and meanwhile guarantees an optimum balancing. From an analytical point of view, this can be summarized in a couple of formulas. If we indicate as S_j the set of subdomains assigned to processor j and indicate with $\sum W_{Sj}$ the total computational weight attained by summing the weights of subdomains belonging to S_j, we can introduce, for each processor j, the value δ_j, defined as

$$\delta_j = \frac{\sum W_{Sj}}{h_j} \tag{5.26}$$

and the final goal is to render all δ_j values nearly equal for all processors. In analytical terms, if δ_{max} and δ_{min} are, respectively, the maximum and minimum value over all processors, the function

$$\frac{\delta_{max} - \delta_{min}}{\delta_{min}} \tag{5.27}$$

must be minimized.

The description is probably clearer if we refer to the example of Figure 5.11, where a generic domain is decomposed with several possible levels of recursion. We assume that the number of processors n_p is four. Both the levels of recursion and the consequent assignment of the attained n_s subdomains to the processors should be optimized. If the four processors were perfectly identical (h_j equal for all j), a possible optimum choice in a balanced bisection could be, for instance, a three-level recursion with two subdomains assigned to each processor (therefore, each set S_j would consists of two elements). In real cases, h_j are rarely identical (even identical processors have, at run time, different load conditions) and the minimization of (5.27) can

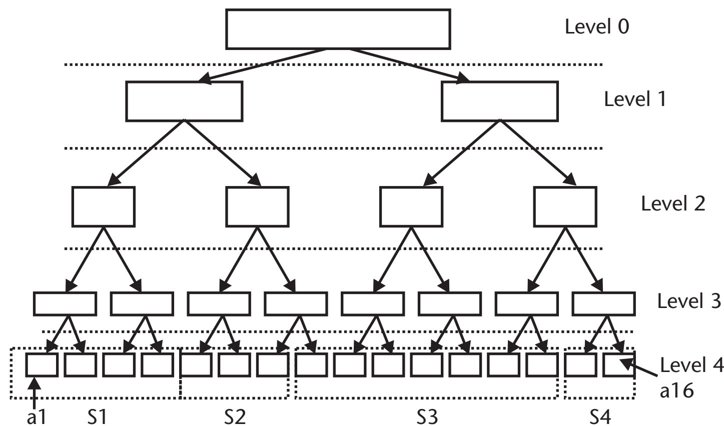

Figure 5.11 The problem of suitable domain partitioning can be solved with a bisection assignment policy. The whole domain can be partitioned into two subdomains and the same operation recursively repeated (balanced or unbalanced partitions can be performed). The example in the figure refers to the case of four processors, with a balanced four-level recursion. It is assumed that each processor has a different weight h_j, so that the attained subsets S_j assigned to each processor have different dimensions: $S_1 = \{a_1, a_2, a_3, a_4\}$, $S_2 = \{a_5, a_6, a_7\}$, $S_3 = \{a_8, a_9, \ldots, a_{14}\}$, and $S_4 = \{a_{15}, a_{16}\}$.

lead, for instance, to the choice of a four-level recursion (consequently $n_s = 16$), with different numbers of subdomains assigned to each processor.

In some cases, the number of recursions can be kept fixed (static decomposition), and the only parameter to be optimized is the partitioning into subsets S_j. Generally, decomposition is dynamic, and the level of recursion can be rearranged at run time. This is the case for our implementation.

5.4.2.2 Implementation Issues

The preliminary analysis (profiling and data flow via Petri nets) allows the exploitation of the previously cited different levels of parallelism. This has three main advantages, each requiring consequent implementation tricks:

1. It allows the achievement of high performance inside each module and AS. This guarantees a high flexibility in the use of the package: one can decide to run in parallel only portions of the package without losing the speed ups achieved for those parts. This compels us to maintain in the parallel implementation a separated structure for each module implementing a MB and the ability to run each MB autonomously (e.g., one can run directly the ERP MB, provided that suitable input data are available via file).
2. The level of parallelism can be tailored for the specific available platform (or platforms). This capability is intrinsically embedded in the load-balancing and scheduling policy described in Section 5.4.2.1 and can also be more explicitly pursued by introducing, during the code implementation, suitable environment variables that allow the dynamic selection of the desired level of parallelism.
3. The approach, as clarified in the previous section, is based on a data-flow philosophy, where software is engineered starting from the analysis of data structures and their evolution from input steps to output processes. This is in accordance with what required by an OO vision of problems (see Chapter 2 for details) and is quite an interesting characteristic in the perspective of a migration towards grid environments. This is a good reason to adopt, as an implementation language, solutions such as Fortran 90, which is open to High-Performance Fortran and is a substantial step towards an OO approach.

As a programming paradigm, message passing is adopted, due to the higher flexibility it guarantees from the algorithmic point of view. The portability is safeguarded by using the MPI standard and running the MPICH implementation, with a look-ahead policy toward GC.

The message-passing paradigm is implemented with the SPMD approach. The choice of SPMD is due to its natural portability toward standard last generation compilers such as the High-Performance Fortran family. Another even more important reason is that the SPMD style is in accordance with the data-flow concept (just as an example, a typical way of conceiving applications with the SPMD approach assumes that input data are scattered among processors before running the application, which implies a dedicated analysis of data demand from each part of the application).

Another crucial implementation issue is the solution adopted for the problem of task composition [27]. Though we have already described how tasks are assigned to processors, we have not clearly defined when they are assigned. To make it simple, the most trivial example of task composition is represented by the serial case (one processor is used). In such a case, a *sequential composition* is adopted: the unit serially composes (processes) tasks. This philosophy can be extended to simple parallel cases, where each processing unit executes serially a certain sequence of tasks, and data differs from unit to unit. This approach is typical for data parallel applications (one extreme example being represented by vector operations) and is naturally amenable for SIMD platforms and applications intrinsically conceivable in such a fashion. Unfortunately, sequential composition is not suited to the CAE of aperture arrays. For instance, the analysis with Petri nets demonstrates a potential concurrency among heterogeneous tasks, such as ASs AFS2-AFS7 in Table 5.1 and ASs related to the AMC module (thus with relevant differences both for the nature of the data processed and for the algorithm to be executed). Consequently, the optimum solution is *concurrent composition* of ASs, where different operations can be performed in the same time on different processors. This algorithmic strategy automatically paves the way for the identification of MIMD platforms as the natural environment suitable for the addressed application.

In conclusion, the package is developed with a SPMD programming paradigm, is assumed to run on MIMD platforms, is developed in Fortran 90 and uses MPICH libraries, and adopts a concurrent composition of tasks to support a multilevel parallelism.

5.4.3 Results on MIMD Supercomputing Platforms

The accuracy of the methods proposed in this chapter can be proven with the analysis of an aperture array used in satellite communication. The results, reported in Figure 5.12, demonstrate that the approximation of infinite metallic flange is not a decisive limitation and the accuracy is extremely satisfactory.

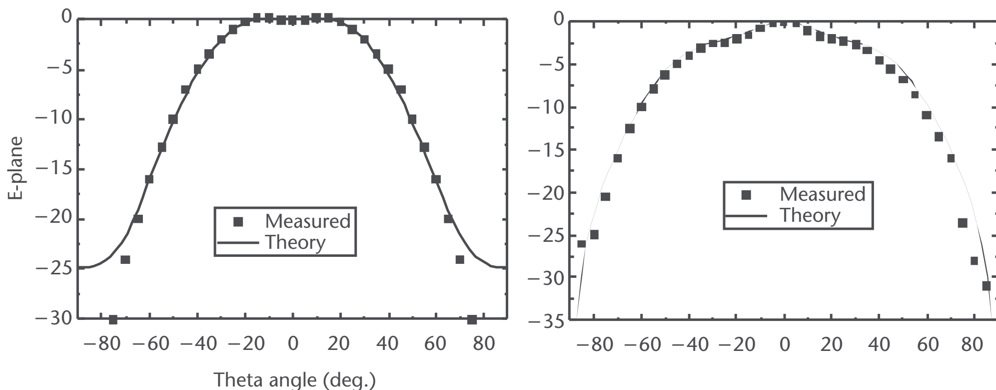

Figure 5.12 Results validating the accuracy of the proposed approach and referring to the array reported in Figure 5.1. We show the copolar radiation patterns in accordance with Ludwig's third definition. Data are referred to the excitation of the central horn, as all other horns are turned off. The working frequency is 3.7 GHz.

Similar conclusions can be drawn referring to other cases simulated by the authors. The only additional observation is that, when higher frequencies are studied (for instance at millimeter-wave frequencies), the finite flange of real systems induces a rippling in the return loss and mutual coupling of apertures, which cannot be numerically simulated with adequate accuracy.

As for the efficiency of the parallel implementation, we report now some results attained on a distributed-memory system, the IBM Scalable Power 2, whose nodes have peak performance of 266 MFLOPS, and an interconnecting device called a high-performance switch (HPS). First, we report in Table 5.3 some speed ups and show the weight of the different MBs of the package. Data refer to a 10-aperture case, analyzed with a 5 × 5–mm spatial resolution and with each horn partitioned into 100 sections. Speed ups are evaluated (as usual) with respect to serial times on the same processor.

Total times in Table 5.3 include some data-scattering and gathering phases for pre- and postprocessing. The reported data refer to the analysis of a single frequency point. The impact of a parallelization on the proposed package is much more interesting if a multifrequency analysis must be performed. Indeed, in such a case, the large majority of the work performed in the first frequency analysis must not be repeated at each iteration (just as an example, consider the evaluation of coupling matrices for each section of each horn or of some integrations in the mutual coupling among apertures). Consequently, the percentage of parallel computation in the successive frequency points substantially increases. To demonstrate this, we propose now some results related to a six-aperture array, with the following characteristics. All horns are identical. Apertures over the flange have the standard dimensions of a WR90 rectangular waveguide (22.86 × 10.16 mm), the sections at the feeding ports of the horns have dimensions 8.5 × 6.1 mm. Horns are 0.6m long and have a flare of 6.02°. Twenty modes are used to expand fields over apertures, and 20 frequency points are studied. Results are reported in Table 5.4.

Table 5.3 Speed-ups for the MBs

Number of Nodes	*Time (s) for MB 1 (AFS)*	*Time (s) for MB 2 (AMC)*	*Time (s) for (MB 3+ MB4) (ESM+ERP)*	*Total Time*	*Speed Up*
2	20.5	375.7	12.9	414	1.85
4	21.3	204.7	12.8	243	3.14
6	21	140	11.6	176	4.35
8	24.4	126.9	11.9	159	4.81

Table 5.4 Speed-ups for a 20-Frequency Case

Number of Nodes	*Total Time (In Seconds)*	*Speed Up*
1	6,547	–
2	3,310	1.97
4	1,706	3.84
6	1,163	5.63
8	872	7.51

As evinced from the reported results, the initial effort to set up all of the data structures and the environment of the parallel implementation is highly advantageous. Data of Table 5.4 refer to the whole package, but similar improvements are observed in each MB, when increasing the number of frequency points.

It is worth observing a couple of things. The first is that the mapping of the MBs and their ASs onto the nodes is adaptive, as it depends on the status of the nodes, in accordance with the domain-decomposition policy described in Section 5.4.2.1. This improves load balancing and maximizes performance.

The second, very relevant, thing is that data reported in Table 5.4 allow us to understand the huge computational effort required by the addressed application, when referred to real industrial cases. When CAE tools are used in the synthesis of aperture arrays, they are very often used in conjunction with optimization tools (as mentioned in Section 5.3.3). In the case of Table 5.4, for instance, the six-aperture array was optimized by using an iterative (genetic) algorithm, which usually needs hundreds of iterations to converge. It is easily noticed from the table that 100 iterations are equivalent to more than 18 hours of computation on a single node. If we recall that we are dealing with only six apertures (a very tiny case!), the criticality of a substantial reduction of computing times for industrial processes is evident. The speed ups attained in Table 5.4 correspond to several working days saved.

5.5 Migration Toward Grid Environments

It is useful, at this point of our analysis, to make a short summary of what we have discussed up to now. As seen in the previous sections, the CAE of rectangular-aperture arrays can be fruitfully partitioned into four main subproblems, namely:

1. AFS;
2. AMC;
3. ESM;
4. ERP.

Each subproblem can correspond to a single, independent module. Each module can in principle be executed independently from the others, provided that suitable input data are available.

Each item needs specific numerical approaches and has peculiar properties and difficulties. Each item requires more or less relative computational effort. Points 2 and 3 are particularly CPU-intensive applications and can take advantage of the use of HPC platforms. It has been demonstrated that the use of a multilevel parallelism, implemented in SPMD programs with MPICH, is a viable pathway to achieve very effective performance on MIMD platforms. In order to maintain the maximum flexibility of the package, a modular structure is still adopted in the parallel implementation, with four independent building blocks.

In accordance with what we've now recalled, the problem of migrating a CAE framework for aperture arrays toward GC is a little more complicated with respect to the case of parallel FDTD (see Chapter 4). In fact, we still need to support an HPC demand, but must fulfill this requirement with an additional and relevant demand

for interoperability among several (namely, four) independent, heterogeneous, and geographically distributed modules.

In Chapter 4, we described how a generic parallel application, developed using MPICH, can be adapted to run on a grid environment. We also showed that computational grids are a very cost-effective way to support HPC. Therefore, in this chapter, we concentrate on the other issues related to cooperative engineering. Before discussing this theme, we want to show the performance attained by migrating the MPICH-based CAE software toward GC. We report now (in Table 5.5) the speed up reached when using Giga-Ethernet in low traffic conditions. We remark that the achieved speed ups are in the worst case 20% smaller than the ones reported in Table 5.4, and attained on a rather costly parallel architecture.

5.5.1 Supporting Cooperative Engineering with GC

Let us assume that the four modules previously enumerated (AFS, AMC, ESM, and ERP) have each been developed by separate and independent working groups, adopting different software and hardware technologies. Let us also suppose that the only constraint for each team is represented by fixed standards for data input and output, so that the four modules can cooperate together via files or via equivalent data communication (e.g., sockets or ftp). We assume that the groups are interested in sharing their applications so that the global task can be performed (CAE of the whole array) but require that their own module remains a proprietary application, resident on their own platforms, with all of the guarantees of security of data and applications. We finally suppose that every team can in principle be interested in "offering" its module for external use at a certain cost.

In conclusion, we have four different modules, distributed geographically throughout the Web, and must guarantee an efficient cooperation among them, with high reliability and security requirements. An environment supporting HPC is also needed, as well as the ability to act as a broker, regulating even aspects such as CPU or application costs and commercial transactions. In the next sections, we describe how GC can comply with these requirements. First, in the next section, we assume that every application accessible through the grid is available for no cost. Later, we shall discuss the problem of adding an economic model in order to allow commercial transactions (e.g., allowing an application's owner to sell its application to a grid member).

Implementation with GC When no economic models are needed, the standard tools inside GT are sufficient. As discussed in Chapters 2 and 3, the GT implements all of the basic services needed to perform the following tasks:

Table 5.5 Speed-ups on a Grid

Number of Nodes	*Speed Up*
1	–
2	1.65
4	3.02
6	4.72
8	6.11

- *Security*. When some groups decide to offer executables for cooperating with other groups, they are free to establish with whom they want to cooperate, when, and how. This requires just some adjustments to GT configuration.
- *RM*. Resources (i.e., executables in our case) can be allocated to each node of the grid, given that permissions are granted.
- *IS*. GT allows a continuous monitoring of the state of resources (CPU, memory, running jobs, and so on) so that load-balancing policies can be pursued.
- *DM*. Data can be moved efficiently in the grid, so that communication among executables can take place.

To describe in more detail how the cooperation can be supported, we consider the grid made up of four nodes and described in Chapter 3. Suppose that each node of the grid hosts a different CAE executable. Suppose also that the executables can communicate with each other via files (i.e., each executable produces a file with data feeding the next executable to run). Then, the GT services allow every user of the grid to perform any of the following operations:

- Choose an executable to run among those dispersed in the grid;
- Select the platform where to run it;
- Install the input file where needed;
- Analyze the output produced by the executable.

Simple scripts containing the needed GT commands can generate the overall CAE simulation by running in sequence the executables and properly moving the input files on the target platforms. In the attached CD-ROM, we provide a script and some C programs simulating this situation. The C programs simulate the dispersed executables and should be installed on different server nodes of the grid. The script simulates the application governing the CAE simulation and should be installed on a client machine of the grid (i.e., on the machine or machines from which users are supposed to launch the simulation).

To recall how this may happen, we resume now the basic commands to be used in a simple, exemplificative case (see Figure 5.13).

Suppose that the executable named "afs" (performing the analysis of the feeding section of the aperture array) is resident on the machine "bach.unile.it" in the directory named "/home/alexa/bin." Suppose also that a user needs to launch the "afs" application from the machine named "picasso.elemgrid.org," where the input file resides. The application "afs" can be launched by creating the following file named "afs.rsl" containing the RSL instructions:

```
&
(count=1)
(jobtype=single)
(directory=/home/alexa/bin)
(executable=gsiftp://bach.unile.it/home/alexa/bin/afs)
```

Count=1 and jobtype=single indicate that the executable must be executed once in a single-threaded fashion, and "/home/alexa/bin" is the path of the directory where the executable must run.

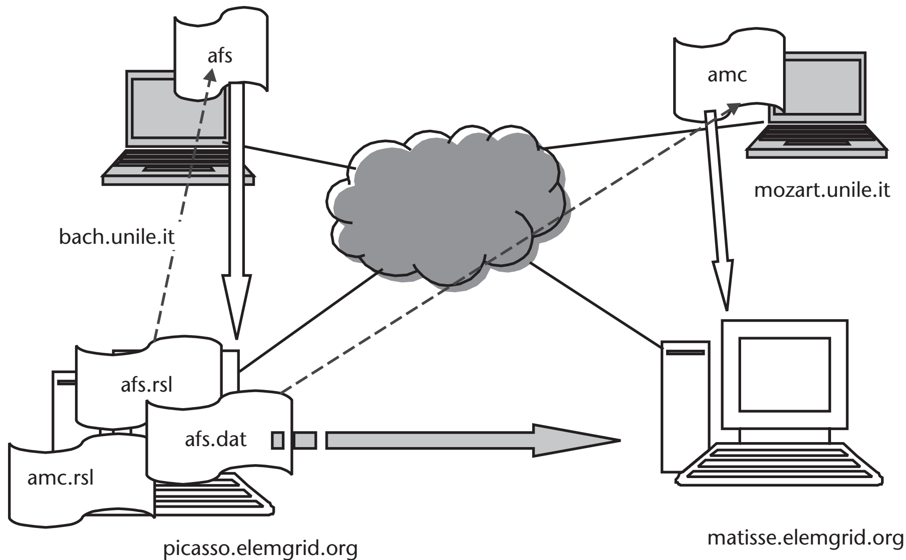

Figure 5.13 GT services allow users to select an application from those available in a grid, choose the platform where to run it, and transfer the input file onto the target machine. The combination of these services allows the integration of application components, thus supporting the cooperation among dispersed research groups. In the figure, a grid made up of four nodes is shown. The user logs into the machine named "picasso.elemgrid.org" and, from there, launches the CAE simulation. He first requests running the executable named "afs" resident on the machine named "bach.unile.it." The application is run via the "globusrun" command, which parses the RSL file named "afs.rsl." In the example, the "afs" executable is asked to run on the local machine, where the output file named "afs.dat" is generated. Then, the user asks to run the executable named "amc" located in the machine named "mozart.unile.it." In case the current less-loaded machine is "picasso.elemgrid.org," the user must first move the input file (gray arrow) there via the "globus-url-copy" command and then run the executable on the target machine.

The line

executable=gsiftp://bach.unile.it/home/alexa/bin/afs

transfers the executable named "afs" from machine "bach.unile.it" onto machine "picasso.elemgrid.org" for a while, so that it can be executed.

The instructions contained in the file named "asf.rsl" can be executed by submitting the following command:

```
globusrun –s –r localhost –f afs.rsl
```

which asks (with the option "–r localhost") to run the executable on the local machine (i.e., the machine named "picasso.elemgrid.org"). Suppose that the executable produces a file, named "afs.dat," whose content feeds the executable named "amc" (performing the analysis of the mutual coupling among apertures). At the end of the execution, the file is located in the execution directory (namely, "/home/alexa/bin/") of the target machine (namely, "picasso.elemgrid.org"). Note that in case of overloading of the target machine, the executable can be launched on any machine in the grid by simply indicating in the command "*globusrun*" the name of a different host and moving the input file onto the same machine, with a procedure similar to that described in the following example.

Suppose now that the user needs to run the executable named “amc” and that the executable named “amc” is located in the machine named “mozart.unile.it.” Suppose also that the currently unloaded machine is “matisse.elemgrid.org.” The user must first transfer the input file named “afs.dat” from machine “picasso.elemgrid.org” to the target machine with the command:

```
globus-url-copy file://home/alexa/bin/afs.dat
gsiftp://matisse.elemgrid.org/home/alexa/bin/afs.dat
```

Then, he must write a RSL file that contains the following lines:

```
&
(count=1)
(jobtype=single)
(directory=/home/alexa/bin)
(executable=gsiftp://mozart.unile.it/home/alexa/bin/amc)
```

and launch the command:

```
globusrun –s –r matisse.elemgrid.org –f amc.rsl
```

where “amc.rsl” is the name of the RSL file.

The description proposed so far has demonstrated that GT allows all of the operations needed to support cooperation among dispersed applications, so that a CAE tool can be conceived as the integration of heterogeneous simulators, dynamically embedded inside a unique framework, depending on the peculiar requirements of the design process. Nonetheless, the pathway for a capillary diffusion of such strategies is not complete until the potential user is given a way to simply interact with the enabling technologies. As a matter of fact, this is one of the main focuses of current research in the area of GC. More specifically, a considerable effort is now being addressed on the development of graphic user-friendly Web applications that, making use of Globus IS services, allow the control of grid resources (hardware and software) via simple *point and click* actions. In other words, in a reasonable scenario for the immediate future, the user can monitor and interact in real time with grid resources via grid-enabled Web browsers. For instance, the choice of a target node will be guided by the graphical interfaces showing the current computational load of each device in the grid and the simple click over an icon will allow the selection of the node. Furthermore, other operations such as file transfer or similar processes will be accomplished as equivalent actions are now performed in graphical user interface–based operating systems.

Nimrod-G We consider now the case when an economic model must be added to the system (i.e., resource owners request the payment of some fee for the usage of their resources). As GT does not support any economy-based computing model, it must be integrated with tools supporting computational economy. These tools have the ability to select target resources based on price, objective, and constraints of users (time or budget being perhaps the most typical).

As seen in Chapter 2, Nimrod-G [28] is a resource-management and scheduling system built on Globus services and freely available on the Internet [29]. It

coordinates access to grid resources via grid middleware services (e.g., Globus). When the application is submitted to the tool for execution, the user can specify the deadline results are needed by and the maximum cost he can support. Grid resources must be listed and communicated to the tool, specifying their attributes, including the cost. Based on such information, Nimrod-G allocates the resources with the goal of optimizing the cost or the application performance, as selected by the user. However, the grid resource availability and load vary over time, so Nimrod-G continuously monitors the state of resources, changing those dedicated to the submitted experiment if it understands that the deadline cannot be met with the current resource set.

Let's go back to our exemplificative grid made up of four nodes. Suppose that CPU usage on grid machines has some cost. Suppose also that the executable "afs" is assumed to be on the local machine and is supposed to run on a remote host. The remote host is automatically chosen by Nimrod-G among a set of machines, which can be indicated by the user in a file called "gatekeeper." In our example, the gatekeeper file may look like:

```
fork      localhost              0.00
globus    matisse.elemgrid.org   0.03
globus    mozart.unile.it        0.04
globus    bach.unile.it          0.02
```

The available machines are the local one (localhost, "picasso.elemgrid.org" in our example) and three remote machines whose FQDNs are "matisse.elemgrid.it," "mozart.elemgrid.it," and "bach.elemgrid.it." Values associated to each machine indicate the cost for a CPU-time unit. This means, for example, that each unit of time on "matisse.elemgrid.org" costs 0.03, while each unit of time on "mozart.elemgrid.it" costs 0.03. The first word of each line expresses the name of the resource manager to be used (the "fork" system calls for the local machine and the "globus" gatekeeper on remote machines).

Once grid resources are listed in the gatekeeper file, the user must describe the application (the input, output, number of iterations, and so on). Nimrod-G allows us to describe the application as being made up of a sequence of steps, each being performed on grid resources. This can be done by preparing a *plan* file. A sample plan file is now reported (and is contained in the attached CD-ROM).

```
# node initialization
parameter frequency label "afs" float range from 4 to 10 step 0.5;
task nodestart
   # copy executable from the root directory to remote node,
   copy afs node:.
   # copy data from the root directory to remote node,
   copy afs.dat node:.
endtask
# individual jobs
task main
      # execute the simulation with corresponding parameter values
      node:execute ./afs  afs.out
      # copy the remote output file to a unique file on the root host
```

```
    copy node:afs.out output.$jobname
endtask
```

Comments begin with the "#" character. In this plan file, the first line:

parameter frequency label "afs" float range from 4 to 10 step 0.5;

asks that executable *afs* is launched several times. More specifically, it asks the *afs* to run for several frequency values, ranging from 4 to 10 GHz, with a step of 0.5 GHz.

The lines included between the "task nodestart" and "endtask" keywords describe operations that must be performed once, before running the executable.

In the example, they are:

copy afs node:.

which asks to copy the executable named "afs" from the original node to a target node; the keyword "node" stands for "remote node chosen by Nimrod-G among the hosts listed in the gatekeeper file," and

copy afs.dat node:.

which asks to copy the file named "afs.dat" from the original node to the remote node.

The lines included between the "task main" and "endtask" keywords describe the operations that must be performed iteratively, for each new value of the parameter, as expressed in the first line of the plan file.

They are:

node:execute ./afs afs.out

which asks to execute the program named "afs" on the remote node (as expressed by the keyword "node") and to redirect the standard output on the file named "afs.out" and the line:

copy node:afs.out output.$jobname

which asks to copy the output file to a local file, whose name is automatically assigned for each job iteration. In this manner, the user will find, at the end of the job, several files named output.$jobname, one for each value of the input parameter.

Once the user has written the gatekeeper file and the plan file, she must launch the job by using the Nimrod-G user interface. Then, Nimrod-G performs the required steps scheduling the resources with the goal of optimizing the cost or the application performance, as selected by the user. At the end of the execution, Nimrod prints out a report specifying total costs.

5.6 Conclusions

In this chapter, we discussed the suitability of GC to support effective cooperative engineering when dealing with the problem of CAE of rectangular-aperture antenna

arrays. The problem, known to be very computationally intensive, requires the use of a multiplicity of numerical techniques to be solved. This can often imply that a CAE tool must integrate heterogeneous software, possibly with a geographical distribution over the Internet, inside a unique framework. The efficient and effective cooperation of the codes must be supported, along with a rigorous respect for security issues and managing with the capability of brokering economical transactions when needed.

It has been proven that GC fulfills these requirements, as well as cost-effective HPC (as already proven in Chapter 4 and confirmed in this chapter). It can be concluded that the migration of complex computational EM applications towards GC can be fruitful in a wide variety of cases, when cooperative engineering, along with parallel computing, can be a candidate pathway to reduce design time and costs and improve the quality of devices.

Acknowledgments

The authors are grateful to Mauro Mongiardo and Cristiano Tomassoni for having substantially contributed to the themes addressed in this chapter.

References

[1] Lewin, L., *Advanced Theory of Waveguides*, London: Iliffe, 1951.

[2] Galejs, J., "Admittance of a Waveguide Radiating into a Stratified Plasma," *IEEE Trans. Antennas Propagat.*, Vol. AP-13, January 1965, pp. 64–70.

[3] Bodnar, D. G. and D. T. Paris, "New Variational Principle in Electromagnetics," *IEEE Trans. Antennas Propagat.*, Vol. AP-18, March 1970, pp. 216–223.

[4] Croswell, W. F., R. C. Ruddock, and D. M. Hatcher, "The Admittance of a Rectangular Waveveguide Radiating into a Dielectric Slab," *IEEE Trans. Antennas Propagat.*, Vol. AP-15, September 1967, pp. 627–633.

[5] Croswell, W. F., et al., "The Input Admittance of a Rectangular Waveveguide-Fed Aperture Under an Inhomogeneous Plasma: Theory and Experiment," *IEEE Trans. Antennas Propagat.*, Vol. AP-16, July 1968, pp. 475–487.

[6] Mailloux, R. J., "Radiation and Near Field Coupling Between Two Collinear Open Ended Waveguides," *IEEE Trans. Antennas Propagat.*, Vol. AP-17, January 1969, pp. 49–55.

[7] Mailloux, R. J., "First Order Solution for Mutual Coupling Between Waveguides Which Propagate Two Orthogonal Modes," *IEEE Trans. Antennas Propagat.*, Vol. AP-17, November 1969, pp. 740–746.

[8] Jamieson, A. R., and T. E. Rozzi, "Rigorous Analysis of Cross Polarization in Flange-Mounted Rectangular Waveguide Radiators," *Electron. Lett.*, Vol. 13, November 24, 1977, pp. 742–744.

[9] Teodoridis, V., T. Sphicopoulos, and F. E. Gardiol, "The Reflection from an Open-Ended Rectangular Waveguide Terminated by a Layered Dielectric Medium," *IEEE Trans. Microwave Theory Tech.,* Vol MTT-36, May 1985, pp. 359–366.

[10] Baudrand, H., J. Tao, and J. Atechian, "Study of Radiating Properties of Open-Ended Rectangular Waveguides," *IEEE Trans. Antennas Propagat.*, Vol. AP-17, August 1988, pp. 1071–1077.

[11] Mongiardo, M., T. Rozzi, "Singular Integral Equation Analysis of Flange-Mounted Rectangular Waveguide Radiators," *IEEE Trans. on Ant. and Prop.*, Vol. 41, May 1993, pp. 556–565.

[12] Bird, T. S., "Mutual Coupling in Finite Coplanar Rectangular Waveguides Arrays," *Electron. Lett.,* Vol. 23, October 1987, pp. 1199–1201.

[13] Bird, T. S., "Analysis of Mutual Coupling in Finite Arrays of Different Sized Rectangular Waveguides," *IEEE Trans. Antennas Propagat.*, Vol. AP-38, February 1990, pp. 166–172.

[14] Bird, T. S., and D. G. Bateman, "Mutual Coupling Between Rotated Horns in a Ground Plane," *IEEE Trans. Antennas Propagat.*, Vol. AP-42, July 1994, pp. 1000–1006.

[15] Kitchener, D., K. Raghavan, and C. G. Parini, "Mutual Coupling in a Finite Planar Array of Rectangular Apertures," *Electronic Letters*, Vol. 23, October 21, 1987, pp. 1169–1170.

[16] Abramowitz, M., and I. Stegun, *Handbook of Mathematical Functions*, New York: Dover Publications, 1974.

[17] Mongiardo, M., L. Tarricone, and C. Tomassoni, "A Comparison of Numerical Methods for the Full-Wave Analysis of Flange Mounted Rectangular Apertures," *Int. Journal Numerical Modelling*, Vol. 13, No. 1, 2000, pp. 21–35.

[18] Collin, R., *Field Theory of Guided Waves*, New York: IEEE Press, 1991.

[19] Conciauro, G., M. Guglielmi, and R. Sorrentino, *Advanced Modal Analysis*, London: Wiley, 1999.

[20] Dobrowolski, J. A., *Introduction to Computational Methods for Microwave Circuit Analysis*, Norwood, MA: Artech House, 1991.

[21] Alessandri F., M. Mongiardo, and R. Sorrentino, "New Efficient Full Wave Optimization of Microwave Circuits by the Adjoint Network Method," *IEEE Microwave Guided Wave Lett.,* Vol. 3, No. 11, November 1993, pp. 414–416.

[22] Mongiardo, M., and R. Ravanelli, "Automated Design of Corrugated Feeds by the Adjoint Network Method," *IEEE Trans. Microwave Theory Tech.,* Vol. 45, May 1997, pp. 787–793.

[23] Collin, R., *Antennas and Radiowave Propagation*, Singapore: Mc Graw-Hill Int. Ed., 1985.

[24] Ludwig, A. C., "The Definition of Cross Polarization," *IEEE Trans. on Ant. and Prop.*, January 1973, pp. 116–118.

[25] http://www.gnu.org.

[26] Petri, C. A., *Kommunikation mit Automaten*, Ph.D. Thesis, University of Bonn, Germany, 1962.

[27] Foster, J., *Designing and Building Parallel Programs*, http://www.netlib.org.

[28] Buyya, R., D. Abramson, and J. Giddy, "Nimrod/G: An Architecture for a Resource Management and Scheduling in a Global Computational Grid," *4th Int. Conf. on High Perf. Comp. in the Asia-Pacific Region,* IEEE Comp. Soc., 2000, pp. 283–289.

[29] http://www.csse.monash.edu.au.

CHAPTER 6

Wireless Radio Base Station Networks

6.1 Introduction

In Chapter 4, we discussed the viability of GC as a low-cost, high-performance computing strategy. An FDTD analysis of human-antenna interaction problems was the field trial. In Chapter 5, we moved towards a more complex and challenging goal, the CAE of aperture-antenna arrays. The development of such an environment casts problems of supercomputing as well as cooperative engineering. It has been demonstrated that GC is an answer suited to both requirements.

In the current chapter, we move toward a third area of application—the design, management, and planning of wireless radio base station (BS) networks. Indeed, the impressive progress in wireless systems and services is compelling operators to reduce the time needed to design the network, improve the quality of service, and better control human exposure to EM fields. This implies a strong demand for automatic tools that optimize the critical parameters (e.g., base station locations, BS power levels, and antenna tilting), allow a rigorous prediction of the fields radiated by base station antennas, and provide an easy interaction with the end user. These goals require a multidisciplinary approach involving the use of radio propagation (RP) models, optimization methods, and sophisticated software technologies. Once again, several needs arise: *supercomputing* strategies, *cooperative engineering* (already addressed in Chapter 4 and 5, respectively), and *real-time data communication and management* in a geographical distributed system.

In this chapter it is shown that GC is the suitable answer to these three needs. As supercomputing and cooperative engineering with GC have already been discussed in other parts of the book, in this chapter the focus is on data communication and management.

The chapter is structured as follows. First, some basic concepts on cellular systems are given. In Section 6.3, we concentrate on some relevant issues for modern wireless systems, which represent a substantial momentum toward automatic tools for network optimum planning. Section 6.4 introduces such tools, and a candidate structure and architecture for a planning tool are proposed in Sections 6.5 and 6.6. The role of GC in this application is described in Section 6.7, while the effective implementation of an optimum planning system with GC is described in Section 6.8, with practical examples focused on data management. Finally, some conclusions are drawn.

6.2 Foundations of Cellular Systems

In this section we propose some very basic principles for the organization of a cellular network, describing its most important characteristics with respect to the power rightsizing of antennas, the traffic management, and coverage. The rapid and continuous evolution of the addressed themes, and above all the spirit of this book, suggest treating them in a very general fashion, focusing on the most fundamental issues, which are probably destined to retain their current arrangement in a rather long-term scenario. Consequently, in this section, we first introduce some general considerations for cellular systems. Later, frequency reuse is described, as well as the relevant concepts of capacity and traffic management. Then, a simple and schematic description of how a wireless system connects users is given, as well as a quick overview on radio-base antennas. More detailed descriptions for all of these subjects, with dedicated analyses for the different generations of systems (second generation GSM networks, third generation UMTS, and so on) can easily be found in the specialized literature [1–3].

6.2.1 General Considerations

A cellular system operates by partitioning the covered area into *cells*, with a rather small radius (macrocells rarely exceed 20 km, with 1–5 km being a much more common distance). The coverage is guaranteed by a network of transmitting and receiving low-power antennas (100W being a rather high reference level), dispersed so that each cell is covered by one installation. The philosophy of a cellular system marks an apparent change with respect to radio-television broadcasting systems, which has a small number of transmitting stations, each covering a very large area. In cellular systems, the capacity to support a large number of users and to increase the coverage is guaranteed by a large number of transmitters with a pervasive spreading. The connection between a wireless cellular network and the traditional wired telephone network is provided by a *switching station*, or hub, which is also in charge of managing several mobile-mobile communication operations. A schematization is proposed in Figure 6.1.

Each cell is assigned a frequency range, and adjacent cells use different frequencies, so that interferences are minimized. The cell size and shape are relevant parameters and are strictly connected with the policy of traffic management. They also influence *capacity* (i.e., the number of sustained users) and coverage. The cell size determines the BS power level. In fact, each BS can be designed with a maximum emitted power level: the available power classes for BSs in the GSM system are reported in Table 6.1.

In principle, the shape of a cell is a free parameter. As a matter of fact, some geometrical forms are more appropriate than others. Indeed, each mobile entity must be assigned to one single cell in each moment, with an easy and unquestionable algorithm. A very simple idea is to assign each mobile to the nearest BS, so that the *path loss* (power loss due to the BS-mobile distance) is minimized. The application of such a principle induces us to consider a hexagonal shape, with the BS positioned in the center. In such a case, all of the adjacent BSs are at the same distance from a reference BS (see Figure 6.2), so that the assignment of a mobile to one BS is simpler.

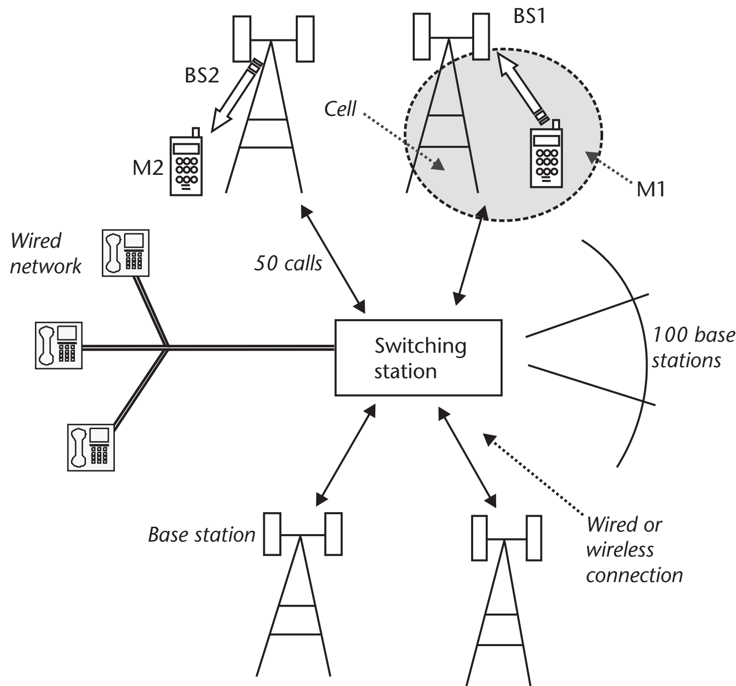

Figure 6.1 The structure of a cellular system. The connection with the wired network is ensured by a hub. A BS typically supports about 50 calls, and the hub can manage about 100 BSs. The hub is also in charge of connecting mobiles belonging to different cells (such as mobiles M1 and M2).

Table 6.1 Available Power Classes for BSs in the GSM System

Power Class	*BS Power (W)*
1	320
2	160
3	80
4	40
5	20
6	10
7	5
8	2.5

Moreover, in order to improve capacity, *sectorization* is a common praxis (i.e., the splitting of channels available in a cell into three subgroups, each covering one-third of the cell area). This implies the use of directional antennas, such as the one we will discuss in Section 6.2.5.

6.2.2 Frequency Reuse

As already mentioned, the allocation of frequencies and channels inside cells must be performed in accordance with the minimization of interference among multiple users and adjacent cells. For instance, the power level of BS antennas, as well as

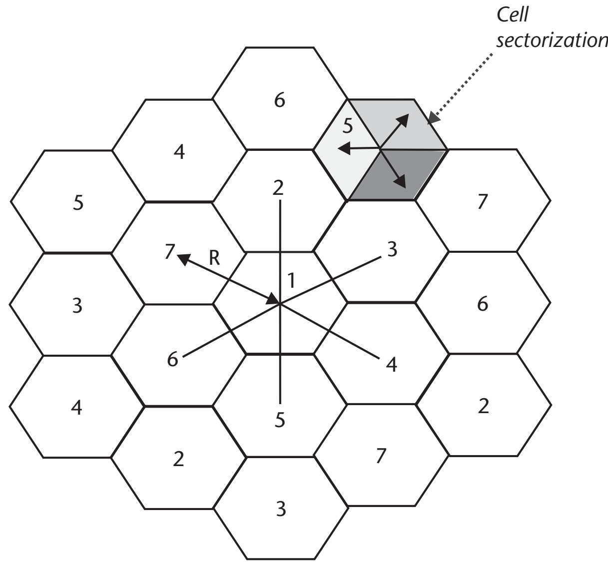

Figure 6.2 Hexagonal cells allow an easy assignment of users to BSs: when BSs are in the cell center, adjacent BSs are at the same distance R from a reference BS. Cells are typically sectorized, so that each sector is assigned a fraction of all of the available channels.

their height, must be tuned so that the field emitted has adequate intensity to cover the pertinent cell. Meanwhile, the field intensity should be as low as possible outside the cell. Of course, this is physically impossible, thus suggesting the use of different frequencies for each cell. Unfortunately, the available frequency band for a system, and the consequent number of channels, is limited. A clever policy of frequency reuse is compulsory. This is attained by introducing the concept of *clustering*. A cluster is a set of cells, each operating on a different frequency and group of channels. In Figure 6.3, some examples of clusters are reported. The cluster dimension N (i.e., the number of cells forming a cluster) is a relevant parameter, which is related to:

- The distance D between the center of two cells using the same frequency;
- The radius r of each cell.

It can be demonstrated that the following relationship holds [2]:

$$\frac{D}{R} = \sqrt{3N} \tag{6.1}$$

For a given number F of frequencies available in the wireless system, the number of channels per cell is F/N. Not all values of N can be adopted, for geometrical reasons. Some feasible N values, frequently encountered, are: 3, 4, 7, 19, and 27. In principle, smaller clusters should guarantee higher spectral efficiency. Unfortunately, low values of N imply higher risks of interferences between cochannel cells (i.e., cells belonging to adjacent clusters and using the same frequency band). This kind of interference is called *first-tier interference*. Consequently, the choice of an appropriate N is critical and complex. Anyway, $N = 7$ is one of the most typical.

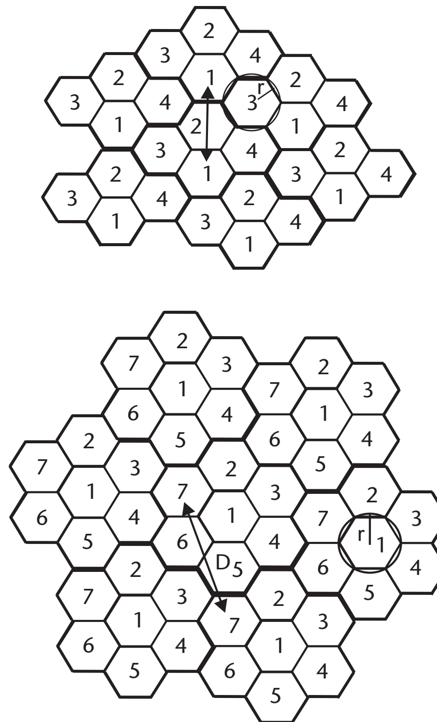

Figure 6.3 Two different cluster topologies with $N = 4$ and $N = 7$.

6.2.3 Capacity and Traffic

Frequency reuse, in many practical cases, is not sufficient to provide an adequate number of channels in a cell. The system, indeed, is designed to sustain an average traffic demand, which is easily exceeded (e.g., in peak hours in urban areas). Specific strategies are introduced to cope with such problems, which can have a severe impact on the quality of service, such as the *blocking* of a call (i.e., its rejection due to the unavailability of channels).

An important strategy (static, involving the permanent reconfiguration of the network) is the introduction of new cells, which can also be attained by partitioning existing cells. For instance, a *macrocell* (radius up to 20 km) can be partitioned into *microcells* (radius up to 1 km) or *picocells* (radius of hundreds of meters). This strategy can also be accompanied by a robust sectorization of cells. All of these artifacts deserve an adequate planning and optimization step, so that the appropriate antennas, with suitable locations, power levels, tilting, and radiation patterns, are selected.

A different and dynamic planning policy is the *frequency reassignment* (i.e., a real-time reconfiguration of channels, so that overloaded cells subtract free channels from idle adjacent cells). This is a complex, yet effective, approach and needs a continuous monitoring of traffic conditions and a proactive management of service requests.

It is worth putting forth that both the static and dynamic policies described here can take substantial advantage of the availability of real-time data on traffic and field

levels at the geographical level [4]. It is also apparent that the reduction of cell size (as well as the dynamic reassignment of channels to cells) renders *handover* critical (handover occurs when a mobile crosses the cell boundary and must change channels).

6.2.4 How a Cellular System Connects Users

In Figure 6.1, we have depicted a schematic representation of how a wireless system is connected with the wired network. Now we describe the generic connection between two users in the wireless network.

As stated earlier, the core of the system is the BS, composed of a receiving and transmitting antenna and a control unit. The control unit connects the BS with the switch. This connection is typically wired; less frequently, it is wireless. The switch is in charge of assigning channels to cells and users, and it manages with handovers.

The mobile unit (e.g., a handheld phone), when working, permanently scans signals emitted by the control unit of each BS. Such signals are needed in order to allow each mobile to identify the nearest BS, so that it connects it. This allows *handshaking* (i.e., the continuous monitoring of the mobile movements inside a cell) or its possible handover. It also worth mentioning that, in some cases, a mobile is assigned to a distant BS due to overloading of the nearest BS.

Thanks to handshaking, every mobile "belongs" to one BS. The mobile is connected with the BS using both a control channel and a traffic channel. When the mobile M1 wants to make a call, it sends a message to its BS (namely, BS1 in Figure 6.1), containing the number of the targeted unit M2. BS1 forwards the request to the switch. The switch broadcasts the request to all of the BSs that can host the target unit M2, and all of the BSs transmit the request to all mobiles in their respective cells (this process is called *paging*). When the target mobile M2 identifies its number in the message received, it sends a reply to its BS (BS2 in Figure 6.1), which notifies this to the switch. The switch assigns a channel to the caller mobile M1 (the assigned channel is one of the available channels in the pertinent BS, namely BS1) and does the same with M2. Meanwhile, it notifies this to the control units of both of the involved BSs, BS1 and BS2. This way, both mobiles M1 and M2 select the assigned channels, and the call is activated. The channels assigned to M1 and M2 are freed when the call is concluded. The call can be broken (*dropped*) when the signal-to-noise ratio is lower than a fixed threshold.

6.2.5 BS Antennas

A typical antenna for a BS is reported in Figure 4.3, along with its radiation pattern in the E and H planes and a representation of the E-field levels in its proximity, calculated with the FDTD code discussed in Chapter 4. Similar antennas are usually adopted for macrocells, which still remain the most common. A more complete review of micro and picocell antennas is found in [2]. As seen in Figure 4.3, the typical BS antenna is made of several half-wave dipoles. This is what we call an *array*: we recall that an array is an interconnection of radiating elements (we assume them all identical) so that a directional pattern is attained [5].

Arrays are linear when the elements are along one line or planar when the elements are in one plane. In BS antennas, the elements can be distributed along one or more columns. Thus, in general we deal with linear or planar arrays. In Figure 4.3, a

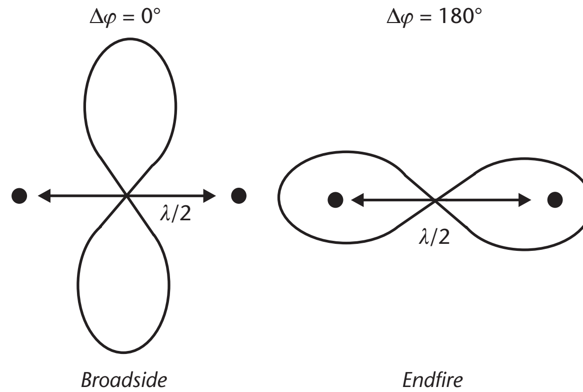

Figure 6.4 An array of two isotropic point sources. The broadside and endfire cases are shown, depending on the phase of radiators.

metal case, which prevents backlobes, can also be observed. Nonetheless, very small backlobes are usually present. The metal case renders the near-field behavior of the array rather too complex to be studied, though the far-field behavior is adherent to the basic theory we are going to recall shortly (we give a very quick refresher to near/far field concept in Appendix C). The use of antenna arrays in BSs helps achieve a strongly directional behavior in the E-plane (this, combined with a suitable mechanical tilting, allows a better exploitation of the available power) and a less marked directivity in the H-plane, so that approximately one-third of a cell is covered, in accordance with the concept of cell sectorization.

6.2.5.1 Antenna Arrays

The behavior of an antenna such as the one reported in Figure 4.3 is easily understood when recalling how an array works. A trivial example can be useful. We consider two isotropic point sources, at a $\lambda/2$ distance (see Figure 6.4). When they have the same phase, they interfere so that the directivity is increased along the symmetry axis. When they are fed in counterphase, directivity is increased along the line passing through the sources.

In the trivial example of Figure 6.4, we see that by using two isotropic elements, a directional pattern is attained. This behavior is what we call the *array factor* (i.e., the parameter that takes into account the interference between radiators).

In order to generalize the concept to more complex arrays of isotropic sources, first we identify a reference element in the array, with current density $J_O(\mathbf{r})$, where $\mathbf{r}$ identifies the position with respect to the system origin. We also assume that all of the elements of the array are similar sources. That is, a generic element has current density:

$$J_n = C_n J_o(\underline{r} - \underline{r}_n) \tag{6.2}$$

with C_n complex. If we indicate with $\mathbf{E}_O(\mathbf{r})$ the field radiated by a single element, the generalization of the empirical discussion previously performed referring to Figure 6.4 is equivalent to affirming that the overall radiated field is:

$$E = FE_o \tag{6.3}$$

where F is the array factor.

The evaluation of F is what we need to predict the radiating behavior of a BS antenna array. In fact, the radiating properties of half-wave dipoles are quite well known, and the following principle of pattern multiplication can be applied [6]:

The pattern of an array of similar sources is the product of the pattern of the individual source (the dipole, in our case) and the pattern of an array of isotropic point sources that have the same locations, amplitudes, and phase as the nonisotropic point sources (this is the array factor).

In other words, the radiation pattern of a linear array of dipoles can be attained by first evaluating its F (estimated by considering the dipoles as isotropic point sources) and then multiplying F by the radiation pattern of a half-wave dipole. The recursive application of such principle is also suited to the case of planar arrays.

BS array antennas are usually fed with the same signal, though the suitable use of phase difference among radiating elements can be considered to achieve electrical tilting, or, when useful, to dynamically reconfigure the radiation pattern. In such cases, we consider *phased* arrays. These solutions are probably going to become the future standard, as will become clear when we discuss adaptive antennas.

6.3 Key Factors for Current and Future Wireless Communications

The astonishing diffusion of wireless communications, and the consequent increase in traffic demand, available services, and coverage render this market one of the most challenging fields for the immediate and medium-term future. Research is continuously looking ahead for new generations of protocols and services, enhancing related radio-frequency technology. We now put forward some issues, though this is not an exhaustive list, but a subjectively chosen selection of key factors for an effective development of wireless networks.

6.3.1 Power Control

Dynamic control of the power emitted by radiating sources in a system is an effective "must." It is highly desirable to receive the signal emitted by the BS at the mobile end (*downlink* connection) with an adequate power level, so that the signal-to-noise ratio at the mobile guarantees a satisfactory quality of communication. On the other hand, the proximity of handheld phones to the user (see Section 6.3.4), together with the limitations of battery charge duration, compel minimization of the mobile emitted power (*uplink* connection), depending on the distance from the BS. This minimization is also useful to reduce the risks of interference among mobiles working with the same frequency (*cochannel* interference).

Power control is implemented with two main techniques, often coexisting in the same system. The first is the so-called *open-cycle* control; the second is the *closed-cycle* one [7]. In the open case, the mobile is in charge of evaluating the BS distance (thanks to a probing signal received from the BS) and automatically regulating its emission (the lower the power received in downlink, the higher the power emitted in

uplink). In the closed case, the BS control unit is in charge of estimating the power level received from a mobile and regulating both the uplink and the downlink power levels.

Power control is also going to become more and more critical in new generations of wireless systems. In fact, a relevant current trend is the diffusion of wireless systems based on spread-spectrum code division multiple access (CDMA), where the transmission is dispersed through a wider bandwidth. In such systems, for reasons of signal-to-noise ratios, it is extremely important that all of the channels work with the same power level, thus avoiding the existence of "dominant" signals, which could cancel weaker channels. In CDMA systems, where all users adopt the same frequency allocation, this is quite critical, thus compelling implementation of a rigorous power control regime.

6.3.2 Managing with More and More Users

Another crucial point is the capability of reconfiguring, statically or dynamically, a system to manage the continuous increase of users and the consequent demands for capacity. One of the most severe limitations is represented by cochannel interference with mobiles using the same frequencies. An emerging technology is represented by *adaptive* antennas. Such devices exploit the basic principles of *phased* arrays (see Section 6.2.5). In a phased array, the C_n complex factor of (6.2) is used to introduce a phase difference among radiators, so that the resulting behavior of the whole array changes, depending on the selection of C_n factor for each element. The development of digital signal processors driving the array elements allows a real-time reconfiguration of C_n parameters. In this way, a BS can direct its radiation (downlink) mainly toward the targeted mobile. This, for reciprocity, also guarantees a high reduction of cochannel interference in the receiving (uplink) connection. Incidentally, both effects also allow a reduction of power in both directions.

The use of such devices can be considered an extreme sectorization of the BS antenna pattern, dedicated to each mobile. The important concomitant effects of interference reduction, with consequent potential decrease of cell size and increased capacity, have a high cost in terms of communication management, BS and mobile design, and network planning. Above all, very deep knowledge on the behavior of the propagation phenomena between mobile and BSs is needed.

6.3.3 System Standardization and Interoperability

The wide variety of cellular systems and providers, the development of new generations of mobiles and protocols, and the proliferation of wireless local area networks (WLANs) with the 802.11 family [1], cast another open problem for the immediate future: the harmonization and interoperability of systems. Roaming among networks (the possibility of working with one carrier/provider even in zones covered by different providers) is still a problem in some geographical systems and an open issue when speaking about WLANs, such as Wi-Fi [8, 9]. Concerted actions are needed among providers and companies involved in the business, both in the design and in the planning and implementation phase.

6.3.4 Concerns in the Public Opinion

The boom in wireless technologies has implied an impressive growth in numbers of mobiles and sprouting of BSs. Meanwhile, in a kind of self-feeding cycle, the attention to and knowledge of potential hazards from human exposure to EM fields have substantially grown. One of the most evident effects is the strong influence public opinion has on any new installation of BS masts. This is not the appropriate place to discuss the details of such an important problem. It is more appropriate to focus on some relevant consequences. First, in several countries, network planning must be performed in accordance with very severe standards and laws limiting the maximum field levels in the environment. Second, it can be extremely important for providers and companies involved in wireless communications to demonstrate concrete intentions and understanding on the subject of exposure control. Third, installing new masts can be difficult. Fourth, as a partial consequence of the previous point, concerted actions could be helpful among providers, so that cositing policies, roaming agreements, and other actions can reduce the impact of networks in a wide sense (e.g., visually and architecturally). Finally, in some countries, local and national authorities have invested relevant amounts of money to set up a network of sensors (typically wideband sensors) dispersed in geographical areas to monitor the level of EM fields, thus collecting data that can be extremely useful when used in an aggregated manner [4].

All five issues confirm the demand for planning policies for wireless network design and management.

6.4 Planning Wireless Networks

The general description proposed in Section 6.2, and the issues addressed in Section 6.3, put forth the growing complexity of cellular network design. The increasing number of users and of available and requested services, the fierce competition among providers for high quality and full coverage, along with more stringent requirements of safety standards for EM human exposure, render the phases of network design and planning of short-, medium-, and long-term development the critical factors for success. Providers and companies involved in such phases have always adopted dedicated and sophisticated software tools for network planning. Their use was often confined to very specific actions, such as the identification of electrical parameters for BS antennas (e.g., mechanical and electrical tilting or direction of the radiation pattern).

It is more and more apparent that the use of such tools is destined to be much more capillary. It is also becoming crucial for network optimization, monitoring, and real-time reconfiguration, thus leading to their substantial rethinking [10]. To be schematic, modern tools for wireless network optimum planning:

- Must be able to manage with the requirement of a very homogeneous coverage, ensuring uniform field levels as needed by the emerging CDMA-based systems (a uniform coverage, in an open-cycle power-control, guarantees that the mobile is always working with a minimum emitted power and reduces cochannel interference both at the mobile and at the BS end);

- Must be open to forthcoming adaptive antenna technology, thus embedding sophisticated RP models (i.e., models that, given the basic antenna parameters and a detailed knowledge of the topographic properties of the environment, predict the field distribution);
- Should take into account the possibility of concerting the design of parts of the network among more subjects, to manage with roaming problems or to handle possible cositing conditions for BS antennas belonging to different providers;
- Must take into account the existence of safety standards, which in some countries fix a severe limitation to EM field levels in the environment;
- Must be able to monitor data on EM emissions, so that traffic demand, coverage, and compliance with safety standards are permanently controlled in real-time [4, 10]. This can be performed by communicating with a network of EM sensors with a pervasive distribution, such as the infrastructure already under construction in Italy.

The discussion proposed up to now drives us to two different planning steps. The first is a *static* phase, aiming at optimizing some design parameters that can hardly be changed in the future. The most immediate example is the optimum location of BSs. This procedure, which is also subject to restrictions due to visual impact or architectural constraints, can play a major role both in the immediate efficiency and effectiveness of the network and in its long-term evolution.

The second planning step is *dynamic,* and deals with the wide variety of parameters amenable to a dynamic (in some cases in real-time) reconfiguration. Some examples are the maximum available power in a BS, mechanical and electrical tilting, frequency allocation when needed, and a BS antenna's radiation pattern reconfiguration.

In both steps, and above all in the dynamic one, of paramount importance is the availability of real data on EM emissions over a geographical scale, in order to drive the optimization of all of the mentioned parameters [4, 10]. An effective communication between the network planning and control system and the previously cited sensor network is highly desirable.

6.5 An Integrated System for Optimum Wireless Network Planning

On the basis of what we discussed in the preceding sections, it can be concluded that a system for the optimum planning of wireless networks must embed some essential features:

- Appropriate models for field prediction;
- Efficient optimization tools for the identification of BS locations, power rightsizing, and tuning of electrical and mechanical parameters;
- Adequate graphical environments for user-friendly interfaces;
- Real-time interconnection with an EM sensor network with geographical distribution.

These basic characteristics imply several additional requirements, the most important of which is probably an adequate support for geographical information (topographical references are needed by RP models) as well as the full interaction with a database management system (DBMS) containing all of the information about available BS antennas and other data useful when tuning the network design.

In this framework, we describe now in a very schematic manner a possible architecture of an integrated system for network optimum planning (ISNOP). For the sake of brevity, we omit details on the single components, referring to the reported bibliography for further information.

6.5.1 Overview of the System

A possible schematic representation of an ISNOP is reported in Figure 6.5. The RP module envelops several RP models, supporting field prediction inside a domain whose geographical (topographical) characteristics are fully described—see Geographic Information System (GIS) module—once the characteristics of EM sources (essentially BS antennas) are completely known (see DBMS module). The optimization (OPT) module is a black box, which is in charge of receiving the problem requirements (variables to be optimized, optimization policy) and outputting the optimum choice of the variables. The geographical support is guaranteed by the GIS module. GIS systems are commercial packages storing and retrieving geographical and topographical information, thus allowing a full georeferencing for all of the entities mapped onto a geographical map (in other words, they support digital cartography and relative database actions). Data related to EM sources are filed inside a dedicated DBMS. The DBMS module, therefore, is an archive containing all of the information about the existing EM sources. The DBMS module, supported by the GIS module, stores geographical and EM characteristics of installed BSs, as well as the EM characteristics of all of the candidate sources for new potential installations.

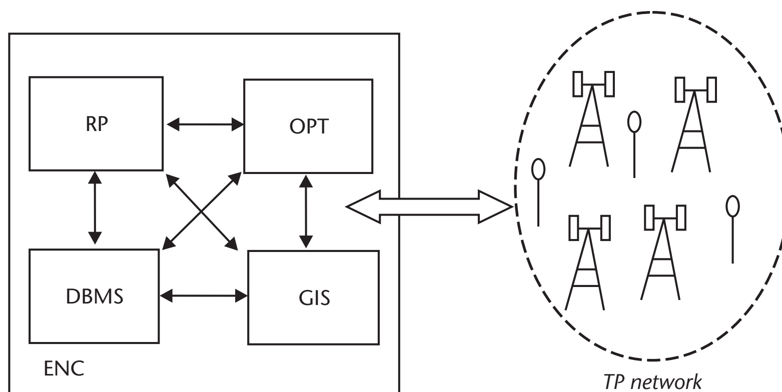

Figure 6.5 A schematic representation of an ISNOP. It consists of four core modules, the RP, OPT, DBMS, and GIS module. The GIS and DBMS modules are in charge of archiving geographical and EM data to form an EM cadastral map. The RP module envelopes different RP models for field prediction, while the OPT module implements one or more optimization algorithms to identify the most suitable value for system parameters. A TPN includes BS antennas and wideband and narrowband field sensors to feed the modules with data. A fifth module, the ENC module, integrates the operation of the core modules together with the TPN.

It is also in charge of filing data related to EM field levels or other relevant information suitable for an *EM cadastral map*.

The enumerated modules can be distributed over the Internet and interact with one another, as well as with a pervasive infrastructure (i.e., the network of EM sensors distributed over the area covered by the wireless system, or at least over some critical parts). It must be observed that in the framework of an ISNOP, the concept of sensors can be generalized. Each point where a sensor is located is intended to be a *test point* (TP). A TP is a point where propagation and service information is available. Consequently, what we have called up to now a "sensor network" must be intended, more generally, as a TP network (TPN). In the TPN, TPs can be both the same BSs and EM wideband or narrowband field sensors.

Finally, the encapsulation (ENC) module is essentially a software entity in charge of supporting the integration of the several modules, their effective interaction with the TPN, and their interoperability.

6.5.1.1 The RP Module

As well described in [2, 11], a wide range of RP models exist. They basically differ from one another by the peculiar application they are customized for, the trade-off between accuracy and computational weight, and the applicability to several classes of problems. As a matter of fact, the ideal model, suitable and preferable for all of the applications, is still a dream. Consequently, the RP module must include a panel of RP models, so that the user can select the most suited model for the specific case.

The most trivial model, yet useful in several situations, is the free-space approximation, based on Friis' formula:

$$E_{eff}\left[V/m\right] = \frac{\sqrt{30 P_T G_T(\vartheta,\varphi)}}{r} \tag{6.4}$$

where P_T is the emitted power, $G_T(\theta,\varphi)$ is the antenna gain in the considered direction, and r is the distance from the antenna electrical center. It is assumed that no obstacles exist between emitter and receiver. This is a very rough prediction, which can strongly overestimate E levels.

When more accurate (though computationally complex) models are needed, candidate alternatives are represented by revisited Okumura-Hata (OH) models [12–16], which are useful when multipath effects can be relevant and line-of-sight (LOS) models such as the free-space approximation are not tolerable. OH models are empirical approaches, which evaluate the effect of the topographical conditions by perturbing the attenuation free-space value with terms that assume different values depending on the height of the BS and of the receiver, on the working frequency, and on the density of buildings.

Another approach is represented by the Walfisch-Ikegami (WI) one [17, 18], revisited by the COST 231 project [19]. In such a model, three attenuation factors are considered: the free-space factor L_F, a factor due to the guiding action of streets (L_{RT}), and a factor due to the diffraction from edges (L_{MS}). Consequently, in the WI model the following formulas are adopted for the attenuation factor L_P in the LOS and nonline-of-sight (NLOS) cases:

$$\text{LOS: } L_P\,[\text{dB}] = 42.6 + 26Log\left(\frac{r}{\text{Km}}\right) + 20Log\left(\frac{f}{\text{MHz}}\right) \tag{6.5}$$

$$\text{NLOS: } L_P\,[\text{dB}] = L_F\,[\text{dB}] + L_{RT}\,[\text{dB}] + L_{MS}\,[\text{dB}] \tag{6.6}$$

where L_{RT} and L_{MS} are estimated with empirical equations.

Finally, in the RP module, fully deterministic approaches should be considered for very critical cases (with small geographical extension). This is the case of ray tracing [2].

6.5.1.2 The OPT Module

Once an RP model is available, and all the characteristics of the BS antennas known, as well as the topography of the domain under investigation, the use of an optimizer is needed in order to identify the best choice for the tunable parameters. This basically implies that:

- An optimization policy is identified, thus leading to an appropriate cost or error function (i.e., a function whose minimization is the final goal);
- An appropriate set of unknowns is identified, whose optimum choice allows the cost minimization.

The definition of both suitable cost functions and optimization unknowns are themselves relevant steps of an optimization procedure.

As quite well known, a generic optimization problem can be solved with a wide variety of approaches, which can be *constructive* (starting from a set of input data, they converge to a solution with a one-shot algorithm) or *iterative* (the convergence is the final result of several iterations, each proposing a candidate intermediate solution). Referring to specialized literature for details [20], we just report here the result of a wide experience on the optimization on BS location, power rightsizing, and mechanical and electrical tilting. For such problems, *Genetic Algorithms* [21] and *Tabu Search* [22] are combinatorial iterative approaches that seem extremely appealing.

These methods, at each iteration, suggest a candidate solution (e.g., the power levels for the BS antennas, their tilting, and their location when new installations must be planned). On such bases, the RP module estimates the field distribution, thus allowing the evaluation of a cost function that includes coverage, field uniformity, compliance with safety standards, and other possible relevant parameters. Depending on the attained cost, a new iteration is started; otherwise, the optimization is halted.

These methods have a high computational cost and must often deal with problems of huge numbers of unknowns. This turns the problem into a computationally intensive one, thus paving the way to the identification of suitable HPC strategies (which are even more necessary when OPT modules are used in conjunction with deterministic approaches in the RP module).

6.5.1.3 The GIS and DBMS Modules

The GIS module essentially consists of an environment with graphical-interface tools supporting digital cartography. An example is shown in Figure 6.6, where field isocurves are reported onto a digitalized map, where every entity corresponds to an addressable element in a database.

The possibility of geo-referring to entities in a GIS allows, for instance, graphical access to the information in the DBMS module. For instance, by clicking onto the dot representing the BS in Figure 6.6, the interface can automatically display all

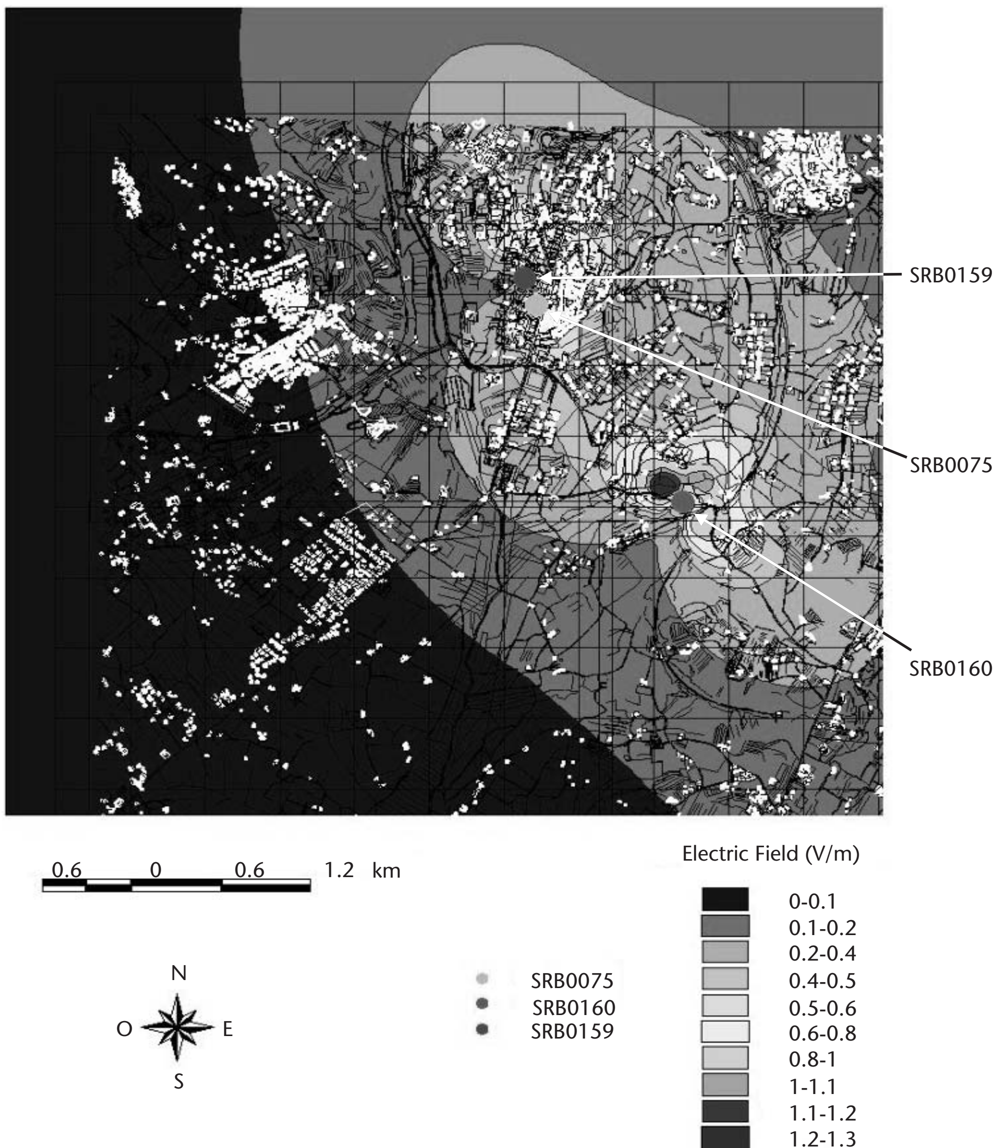

Figure 6.6 Digital map with E-field isocurves. BS antennas are easily identified.

of the relevant data for the BS, as reported in Figure 6.7. The harmonized use of the GIS and DBMS module is a key factor for the effective usability of an ISNOP.

6.5.1.4 The TPN

The TPN plays a major role in an ISNOP. It allows the real-time monitoring of the network behavior, as well as of the service demand over the territory and of compliance with safety standards. As stated before, the same BS antennas are part of this network, as they can provide information both on field levels and on many other key data, such as traffic and service requests. Also wideband or frequency-selective sensors, able to monitor in real-time the electric field, or EM power levels, can belong to the TPN.

In such a framework, TPs produce data with a certain data rate (with data being, for instance, the E field or power level samples). TPs can also receive data. For instance, when a TP corresponds to a BS antenna, it can receive data from the other components of the ISNOP in order to reconfigure itself and improve the wireless network performance.

6.5.1.5 The ENC module

It is apparent that a core position in the ISNOP is occupied by the ENC module, which is involved in guaranteeing the interaction among the dispersed components of the system. It is a sort of universal glue, sticking together the several modules and

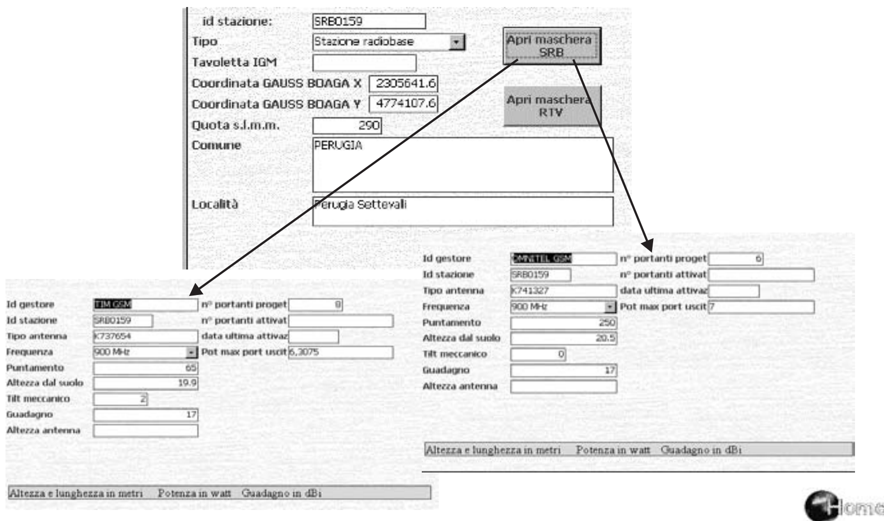

Figure 6.7 The DBMS module represents an EM cadastral archive. In the figure, data about a BS installation are reported, as displayed when accessing the DBMS module. The picture is a snapshot from the original ISNOP interface (in Italian). Labels in the upper mask have the following English translation: "station id," "station class," "Gauss-Boaga coordinates," "height above sea level," "city," and "city district." The push-button from which the arrow starts has the following label: "Open radio base station masks." Labels in the lower masks have the following English translation (from top left to bottom right): "provider," "station," "antenna type," "working frequency," "maximum gain direction," "height from ground," "mechanical tilting," "gain," "antenna height," "maximum number of channels," "number of activated channels," "date of last activation," and "maximum power for channel."

smoothing the technological discrepancies among components, thus driving toward interoperability and compatibility.

6.6 A Candidate Architecture for an Effective ISNOP

In Section 6.5 we have proposed a schematization of the ISNOP structure. Now, we identify a possible architecture supporting it. First of all, it is worth noting that:

1. An ISNOP is intrinsically a wide area distributed system. Indeed, TPs are geographically distributed and the several modules could be dispersed: they could be developed by different producers and could be resident on their own platforms.
2. The several modules can be developed with heterogeneous technologies and methodologies.
3. It is recommended that different implementations are available for the same module. For instance, it could be extremely important to select among different RP modules, resident on different platforms (part of the distributed system), implementing different RP models.
4. Data security is a key factor, especially considering the mission criticality of the system and the stringent security requirements for data concerning traffic and coverage.
5. The system has a high demand for computational power: some RP models and the OPT module require huge computational efforts.
6. Data communication and management among TPs and the remaining components of the ISNOP is critical.

How can these requirements be satisfied? A distributed, Internet-based system is the first, immediate framework inside which a low-cost and simple answer is tentatively found. This distributed environment:

a. Glues TPs to one another and interfaces them with all other components of the ISNOP, so that data produced by TPs can feed the overall computing steps and data returned by the ISNOP can suitably drive the wireless network;
b. Selects, at each moment, which module must be activated, where it is, and where it must run (a brokering function);
c. Provides all of the computational power needed, gathering it together dynamically depending on the current availability and demand.

From a hardware point of view, TPs in an ISNOP must be interfaced via a serial port (or equivalent interfaces) with a *local node* (typically a low-/medium-level PC). The distributed environment can include other nodes (i.e., computers not interfaced with sensors or BS antennas), which can provide, for instance, more intensive computational capabilities, or can "simply" host the ISNOP modules (e.g., RP or OPT). In the distributed system now proposed, the existence of one or more nodes delegated to govern local node activity and interactions must be scheduled and assumes a paramount importance. Such "governing" nodes are called *brokering nodes*.

Brokering nodes must manage with data provided by TPs, thus coordinating the action of the dispersed single modules. They must also communicate to BSs (via local nodes) the result of the computation. For instance, when an optimization of the power levels of BS antennas is performed, they reconfigure BSs in accordance with what is suggested by the OPT module.

6.7 GC and Its Role in the ISNOP

Now, recalling the description of computational grids in Chapter 2, it can be observed that all of the characteristics summarized in Section 6.6 are fully satisfied by GC. More specifically, Chapter 4 has already demonstrated that HPC is affordably sustained and that dynamic HPC architectures can be arranged, as required in points 5 and c of the previous section. On the other hand, in Chapter 5 it was proven that the brokering functions can be easily embedded, thus answering questions arisen in points 3 and b.

Furthermore, points 1 and 2 are intrinsically satisfied, as GC is a more general view of distributed systems with a special focus on security issues, thus fulfilling point 4 as well.

With respect to the lists of points reported in Section 6.6, it can be noted that issues 6 and a are still pending. They are related to data communication and management, and it is demonstrated in the following paragraphs that GC is the appropriate answer to these questions as well.

Now, it can be useful to propose a synthetic scheme representing a possible architecture of an ISNOP implemented on a grid-based distributed system. In Figure 6.8 a schematic representation is reported, as well as an exemplification of the dispersion of components among the ISNOP. It is observed that TPs are connected to local nodes (PCs), and modules are resident on local nodes or on the brokering node. All of the nodes host the grid middleware software (in our case, GT) and are Internet connected. It must be put forth that a special role is played by the ENC module, which is in charge of integrating all of the software components of the ISNOP, dynamically embedding the parts needed in each moment, and freeing modules not needed any longer. Due to its nature and goal, the ENC module, which basically has brokering functions, is assumed to be resident only on brokering nodes. In conclusion, the ENC module can be viewed as a *shell* application, which invokes the main GT commands (i.e., commands described in Chapter 4 and 5) plus the ones we are going to describe in the following.

6.8 Wireless Network Planning with GC

In Section 6.7 we have described a complex infrastructure, based on GC, supporting an ISNOP and all its functions. As is apparent from Figure 6.8, we deal with a large architecture, whose design, development, and management cast difficult and multidisciplinary problems, some of which have already been addressed in Chapters 4 and 5. In this chapter, we want to focus on those problems related to data communication and management, addressing Chapters 4 and 5 for solutions to the remaining ones.

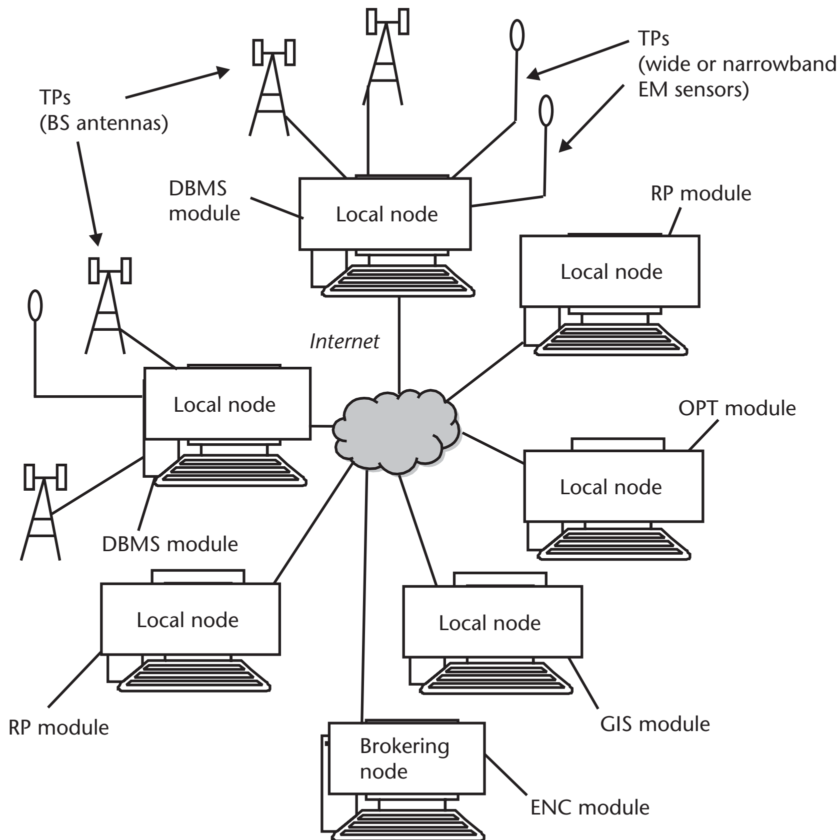

Figure 6.8 The TPN and the dispersion of components through the ISNOP. Local nodes are interfaced with TPs (BS antennas and EM sensors) and host an ISNOP module. Special nodes, called *brokering nodes*, govern local node activity and interaction. The ENC module resides on brokering nodes.

In order to isolate the discussion on data communication in an ISNOP and to facilitate the description of the viable solutions via GC, we consider in the following a simplified version of the ISNOP, as reported in Figure 6.9 (a detailed description of the "simplified" ISNOP follows in a while).

The simplified ISNOP can be introduced if we suppose that, for instance, all of the executables (one instance for each module) are resident on the same node and that the node resources satisfy all current computing demands. This is not a strong limitation: we saw in the previous chapters that grid technology allows both the parallelization of code and the cooperation of applications, and in both cases it gives to the user the perception of interacting with a single high performing machine/application.

We introduce another little simplification: to focus on the movement of single data, we suppose that each TP (i.e., a BS antenna or an EM sensor) is interfaced with its own local node (this means that each TP is connected with a PC; for instance, via a serial port).

Furthermore, we assume that a TP has a different behavior, depending on whether it is an EM sensor or a BS antenna. In fact, though BS antennas and EM sensors are both in charge of measuring EM field/power levels and traffic and of

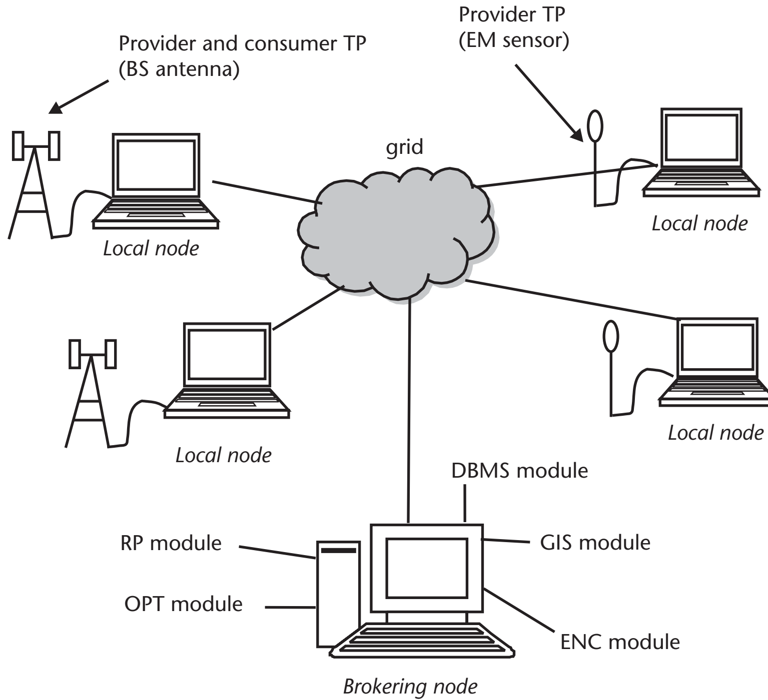

Figure 6.9 The simplified ISNOP is based on a number of assumptions: each module contains only one application; the five modules (executables) are resident on the same node (the *brokering node*); and each TP is interfaced with a different *local node.* Moreover, the model distinguishes between provider TPs (i.e., TPs outputting data) and consumer TPs (i.e., TPs accepting data as input).

sending them to the suitable ISNOP modules, BS antennas can also receive data from the rest of the ISNOP (e.g., when their configuration must be modified). This is not true for EM sensors, which only transmit data. Now, if we call "data provider" an entity in charge of feeding parts of the ISNOP with data, and "data consumer" an entity that receives data from parts of the ISNOP, BS antennas are both *data providers* and *consumers*, while sensors can be considered only *data providers*. BS antennas provide data related to EM fields and traffic. They also receive data (e.g., those needed to modify their power levels and tilting and attained by the OPT module of the ISNOP).

Coherently with the sketched situation, in the simplified ISNOP we have *provider TPs* (PTPs) and *consumer TPs* (CTPs).

In summary, we have:

- A network of PTPs, each interfaced with a local node;
- A network of CTPs, each interfaced with a local node;
- A brokering node hosting both the ENC module and the ISNOP remaining modules (i.e., GIS, DBMS, RP, and OPT).

A double flux of data is present: from the PTPs toward the brokering node (see Figure 6.10) and from the brokering node toward the CTPs (see Figure 6.11). The ENC module is responsible for invoking the proper module to work out input data and to return output data to BSs.

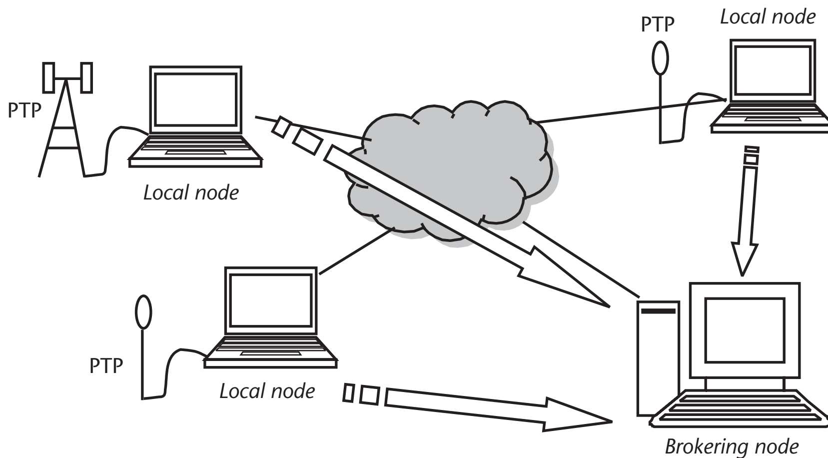

Figure 6.10 A grid of PTPs feeds the brokering node with data.

In the following subsections, the code to simulate such a system is described. In the attached CD-ROM, we included the C code and the shell scripts needed to implement it.

6.8.1 Data Communication with GC in a Simplified ISNOP

Before going on, it is worth putting forth that the following discussion deals with *real-time* data communication and management. While a plethora of alternative techniques could have been adopted for off-line data management (e.g., ftp), GC is *the* solution for an effective real-time approach.

Coming to the implementation of data communication with GC, we introduce a further simplification: we suppose that data produced by the PTPs are written onto files. Moreover, as data must be elaborated by modules resident on the brokering node, the most pertinent location for files produced by the PTPs is the brokering node itself. We can then sketch the following scenario:

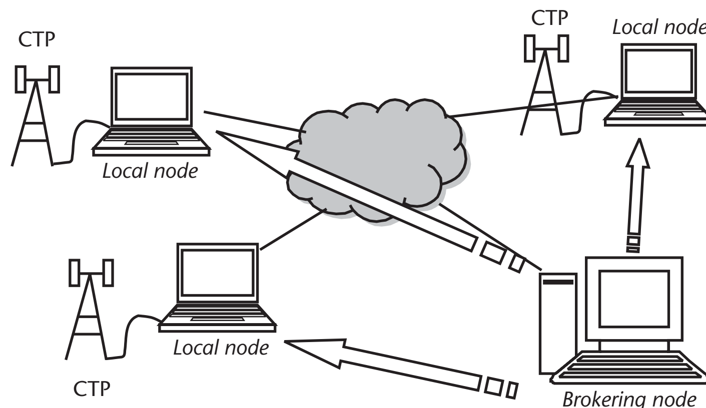

Figure 6.11 Applications resident on the brokering node work out data produced by the PTPs. The results are then scattered to the CTPs.

- Each PTP updates regularly its own file resident on the brokering machine.
- A simple application resident on the brokering machine extracts data coming from different PTPs relative to the same instant.
- An ISNOP module elaborates data extracted at the previous step and generates control parameters.
- Control parameters are scattered to the CTPs.

We explain now how to develop an application that is resident on each local node interfaced with a PTP and can update files resident in the brokering machine. Each PTP updates its own file. The section explains the following core issues:

- How PTPs can update remote files;
- How files must be named so that they are uniquely identifiable in the network;
- Which contents the files must include.

6.8.1.1 Remote File Access

The GT GASS file access API (see Chapter 3, Section 3.9) allows applications to write, read, or modify remote files. This is very useful for our purpose: thanks to the GT GASS file access API, the PTPs can update the brokering archive in real time.

As explained in Chapter 3, GT GASS API allows users to read and write onto remote files with ordinary UNIX I/O and C calls, given that the Globus calls are used in place of open(), close(), fopen(), and fclose(), typical of C and UNIX. More in detail, to elaborate on what already explained in Chapter 3, we recall that in order to update a remote file, a C application must include the following code:

- Activation of the Globus GASS File API module:

  globus_module_activate(GLOBUS_GASS_FILE_MODULE);

- File open via the Globus call:

  fd = globus_gass_fopen("http://hostname:port/filename", ...);

- Ordinary C and UNIX read and write functions:

  fprintf(fd,…)

- File close via the Globus call:

  globus_gass_fclose(fd);

- GASS deactivation via the GASS function:

  globus_module_deactivate(GLOBUS_GASS_FILE_MODULE);

How these calls are used in the context of our application is explained shortly, after discussing how to name the remote files and which data must be written onto.

6.8.1.2 Remote File Naming

We give now some details about the way remote files are named. The "globus_gass_fopen" function addresses a remote file by specifying the protocol, address, and port of the remote server application. These are returned by the GASS server application when it is started on the server machine. Suppose that the broker node is named "picasso.elemgrid.org" and that the GASS server is started via the command:

```
globus-gass-server  -r -w -p3003  &
```

The command returns the string "https://picasso.elemgrid.org:3003," which identifies the server GASS server application. This string must be completed with the name of the remote file.

To distinguish between files produced by different PTPs, we use the FQDN associated with the local nodes with which they are interfaced, as a unique identifier allows us to distinguish between nodes in a network. This means that we assume that each file is named with the FQDN of the machine that produces it. To do this, a couple of useful C functions are adopted. The following code calls the C functions gethostname() and getdomainname() to get information about the machine identification and builds the name of the file using these data (variable named "url"):

```
char name[50],domain[50],url[250];
gethostname(name,50);
getdomainname(domain,50);
sprintf(url,"https://picasso.elemgrid.org:3003/home/alexa/ISNOP/data/%s.%s",name,
domain);
```

This is a portable code that can be installed without any modification on each new node added to the TPN. It queries the FQDN of the machine where it runs by calling the functions gethostname() and getdomainname() so that the file can be named with the strings returned by these functions (variables "name" and "domain") (see Figure 6.12). The code stores in the variable named "url" the string needed to access the remote file and contains the protocol ("https"), the address of the target machine ("picasso.elemgrid.org"), the port ("3003"), the path ("/home/alexa/ISNOP/data") where the file is located, and the filename (contents of the variable "name.domain") needed to access the remote file.

For example, if the client machine has the FQDN "mozart.unile.it," the variable "url" contains the string:

https://picasso.elemgrid.org:3003/home/alexa/ISNOP/data/mozart.unile.it

6.8.1.3 File Contents

Each PTP feeds its own file with data and timestamps. It is important that each datum is accompanied with the timestamp of the time when it was produced, in order to cope with network delays, which provoke a discrepancy between the time that data are generated and the time that data are effectively written onto the file. Timestamps must be congruent (i.e., they must refer to a common clock); otherwise,

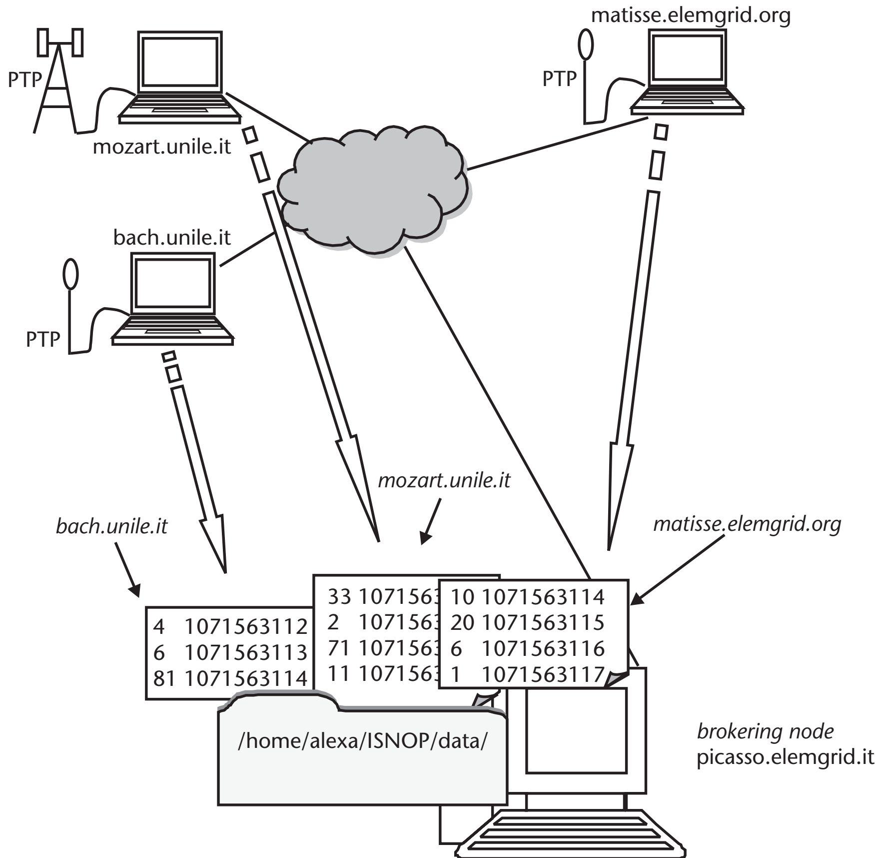

Figure 6.12 A common directory located in the brokering node (namely "/home/alexa/ISNOP/data/") is populated with files updated by the PTPs. Files are unambiguously distinguishable as they are named with the FQDN of the machine they are produced by and contain data and timestamps.

the server cannot work on data truly referring to the same instant. The NTP is the Internet standard to synchronize dispersed machines with a common reference clock. An implementation of the NTP protocol is the *ntp* distribution [23], freely distributed in Internet. Each machine involved in the grid must host the ntp distribution and ntp must be configured so that the machines share the same time clock. This is done by electing one or more time servers and requesting that ntp synchronizes the time of the machine to the common time servers. The synchronization is performed by means of message exchanges with the correct timestamps and the calculation of errors due to network delay times.

Now we give some details on how to install and configure ntp.

Login as root and create the directory where you want to install the package (e.g., "/usr/local/ntp"):

```
mkdir /usr/local/ntp
```

Once the package has been downloaded, extract the files from it:

```
gzip ntp-4.1.1btar.gz -dc | tar xf -
```

This command creates a directory named ntp-4.1.1b. Move to that directory:

```
cd ntp-4.1.1b
```

Update the configuration data:

```
./configure
```

Compile:

```
make
```

Install the executables at the default directory (named "/usr/local/ntp/"):

```
make install
```

Once ntp has been installed, a simple ASCII configuration file must be created. It is named "/etc/ntp.conf" and includes the name of the clock server with which to synchronize. The ntp server must then be started:

```
ntpd &
```

Once the machines are synchronized, the simulation code can run on each PTP machine. In the following, the overall code for the simulation of the PTP grid is included. A simple loop simulates the continuous generation of data, while the call to the C function "sleep" is used to simulate the delay between different data generation (the time step is supposed to be 1s). The C function named "time" returns the time since the "Epoch" (00:00:00, January 1, 1970), expressed in seconds.

```
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include "globus_common.h"
#include "globus_gass_file.h"
#include "globus_error.h"
main()
{
    FILE *fp;
    int rc=0;
    int i=0;
    char name[50], domain[50], url[250];
    time_t t;
    if((rc=globus_module_activate(GLOBUS_GASS_FILE_MODULE))
           !=GLOBUS_SUCCESS)
{
    printf("gass activation failed\n"); exit(-1);
}
    gethostname(name,50);
    getdomainname(domain,50);
```

```
        sprintf(url,"https://picasso.elem
        grid.org:3003/home/alexa/ISNOP/data/%s.%s",name,domain);
    for(;;)
    {
        fp=globus_gass_fopen(url,"a");
        time(&t);
        rc=fprintf(fp,"%d\t %d\n",t,i);
        sleep(1);
        globus_gass_fclose(fp);
        i++;
    }
        globus_module_deactivate(GLOBUS_GASS_FILE_MODULE);
    }
```

6.8.2 ENC Module Simulation

The brokering machine hosts, in the same directory, a number of files (as many as the PTPs), each named with the FQDN of the feeding local node and containing data and timestamps. A simple shell script can extract data relative to the same timestamp so that they can be worked out (typically by the OPT and RP modules) (see Figure 6.13).

In Figure 6.14, we show the flow chart of the shell script, whose code is included in the attached CD-ROM.

Substantially, the script performs the following tasks:

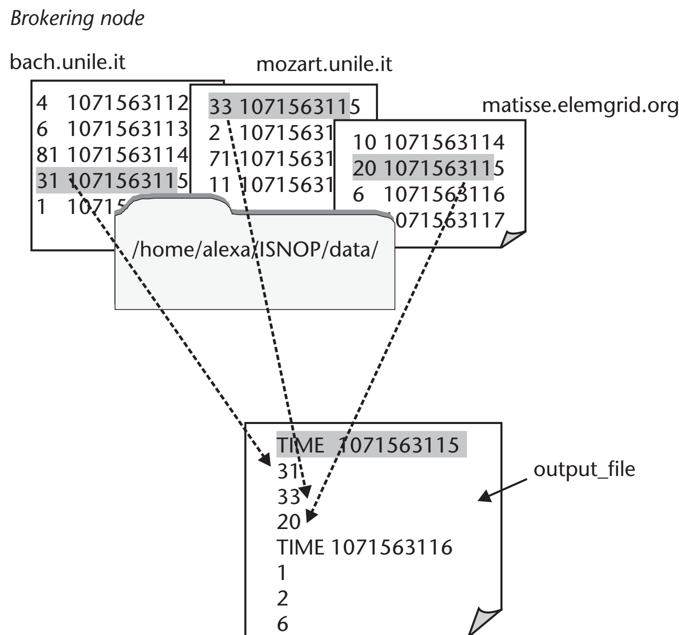

Figure 6.13 Data written by the dispersed PTPs must be gathered so that values relative to the same timestamp can be worked out together. This can be done by a simple application that scans the contents of the directory hosting the data files and produces a new file with grouped data (namely "output_file").

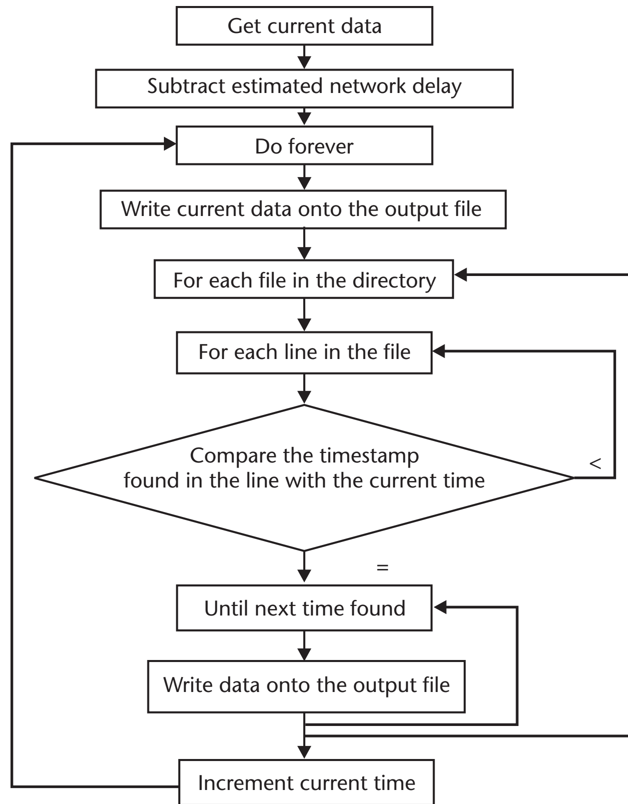

Figure 6.14 Flow chart of the application in charge of rearranging data stored in multiple files by the dispersed PTPs. The application produces a single file where data are grouped based on the timestamps.

- It scans the directory hosting the data files of the PTPs.
- It scans the lines contained in each file looking for the timestamp matching the current time.
- It produces a file containing the timestamps, each followed by the list of data related to it.

To further clarify, we include below the commented code written in the UNIX bash shell language (see Appendix A).

```
/*query the current time and store it in the variable named "curtime"*/
curtime=$(date +%s)
/*subtract some seconds to the current data to take into account the delay due to data transfer*/
let curtime=$curtime-5
/* forever do*/
while true do
/* write the current time onto the output file */
echo "TIME $curtime" > output_file;
/* for each file included in the data directory*/
for i in * do
```

```
/* for each line of the file */
   while read line do
/* parse the current line and put the first field (datum) in the first positional variable
($1)  and the second field (timestamp) in the second positional variable ($2) */
   set – $(echo $line);
   /* compare the timestamp ($2) with the current time ($curtime)*/
   if($2==$curtime) then
      /*write the datum onto the output file*/
   echo $1 > output_file;
   fi
done $i /*end of while loop */
done /* endfor*/
/*increment the current time*/
   let curtime=$curtime+1;
done /*end of forever loop */
```

The output is a file containing, for each timestamp, the list of data related to it. This file can be the input of an ISNOP module (e.g., the OPT module). Once the module produces its output, the output data must be scattered to the dispersed BS antennas (CTPs). This can be done with an application using the GASS file access API, in a fashion similar to that described in Section 6.8.1.1. Separate from the application therein described, where we have a many-to-one communication (from the PTPs toward the brokering node), now we deal with a simpler case: one-to-many communication (from the brokering node to the CTPs). The application can scatter data to the PTPs' antennas once it has a list of the target PTPs. Such a list can be dynamically updated and queried by introducing a *subscription* mechanism, so that each PTP, when added to the grid, can notify its existence to the system by directly accessing a shared archive and by communicating to it its identity and the parameters it is interested in. This can be implemented by using GT services, which include the GASS file access API.

6.9 Conclusions

The area of wireless communications, and more specifically the optimum planning of wireless networks, is one of the most promising for the immediate and medium-term future. The problems related to the development of integrated systems for automatic optimum network planning are multifolded, multidisciplinary, and complex. They gather HPC requirements, system integration, and cooperative engineering issues, as well as effective real-time data communication and management.

We have demonstrated that all of these areas are covered with the same strategy: GC. On the top of the discussions performed in Chapter 4—GC to support HPC—and in Chapter 5—GC for cooperative engineering (CE)—we have here described how to use GC to support data communication and management along with both HPC and CE. In other words, it has been demonstrated that GC is, in one shot, the answer to all three (complex) demands, thus opening interesting perspectives for many other possible applications.

Acknowledgments

The authors are grateful to Maila Strappini, who provided impressive momentum to the beginning of the ISNOP project and still is a pillar for the team working on the project. Many thanks to Federico Malucelli and Maddalena Nonato for stimulating discussions and to Beatrice Di Chiara for her contribution to the development of research.

References

[1] Stalling, W., *Wireless Communications and Networks*, Englewood Cliffs, NJ: Prentice Hall, 2002.

[2] Saunders, S. R., *Antennas and Propagation for Wireless Communication Systems*, London: Wiley, 1999.

[3] Siziak, K., *Radiowave Propagation and Antennas for Personal Communications*, Norwood, MA: Artech House, 1998.

[4] Collmann, R. R., "Evaluation of Methods for Determining the Mobile Traffic Distribution in Cellular Radio Networks," *IEEE Transactions on Vehicular Technology,* November 2001, Vol. 50, No. 6, pp. 1629–1635.

[5] Stutzman, W. L., and G. A. Thiele, *Antenna Theory and Design*, New York: Wiley, 1981.

[6] Kraus, J. D., and R. J. Marefka, *Antennas for all Applications*, New York: McGraw Hill, 2002.

[7] Gibson, J., *The Communication Handbook*, Boca Raton, FL: CRC Press, 1997.

[8] Vaughan-Nichols, S. J., "The Challenge of Wi-Fi Roaming," *Computer,* July 2003, pp. 17–19.

[9] Blau, J., "Wi-Fi Hotspot Networks Sprout like Mushrooms," *IEEE Spectrum*, September 2002, pp. 18–20.

[10] Hurley, S., "Planning Effective Cellular Mobile Radio Networks," *IEEE Transactions on Vehicular Technology*, Vol. 51, No. 2, 2002, pp. 243–253.

[11] Bertoni, H., *Radio Propagation for Modern Wireless Systems*, Englewood Cliffs, NJ: Prentice Hall, 2000.

[12] Okumura,Y., et al., "Field Strength and its Variability in VHF and UHF Land Mobile Service," *in Review of the Electrical Communication Laboratory*, Vol. 16, No. 9–10, September–October 1968.

[13] Hata, M., "Empirical Formula for Propagation Loss in Land Mobile Radio Services," *IEEE Transactions on Vehicular Technology*, Vol. VT-29, No. 3, August 1980.

[14] Damosso, E., "Digital Mobile Radio: COST 231 View on the Evolution Towards 3rd Generation Systems," *Final Report of the COST 231 Project,* European Commission, Brussels, 1998.

[15] http://www.lx.it.pt/cost231.

[16] Kürner, T. (E-Plus Mobilfunk GmbH, Germany), "Propagation Models for Macro-Cells," in *Final Report of the COST 231 Project,* Cap. 4.4, European Commission, Brussels, 1998.

[17] Walfisch, J., and H. L. Bertoni, " A Theoretical Model of UHF Propagation in Urban Environments," *IEEE Transaction on Antennas and Propagation*, Vol. 36, No. 12, December 1988.

[18] Igekami, F., et al., "Propagation Factors Controlling Mean Field Strength on Urban Streets," *IEEE Transaction on Antennas and Propagation*, Vol. AP-26, No. 8, August 1984.

[19] Lähteenmäki, J. (VTT Information Technology, Finland), "Indoor Propagation Models," in *Final Report of the COST 231 Project,* Cap. 4.7, European Commission, Brussels, 1998.

[20] Reeves, C. R. (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Oxford, UK: Blackwell Scientific Press, 1992.

[21] Goldberg, D. E., *Genetic Algorithm in Search, Optimization and Machine-Learning*, Reading, MA: Addison Wesley, 1992.

[22] Glover, F., and M. Laguna, *Tabu Search*, Norwell, MA: Kluwer, 1997.

[23] http://www.ntp.org.

CHAPTER 7

Conclusions and Future Trends

7.1 GC: Benefits and Limitations

In this book we have introduced the use of GC to solve EM problems. After recalling basic concepts on parallel and distributed computing, as well as on Web technologies, in Chapter 1 foundations for GC are proposed. Chapter 2 describes the candidate technologies to support GC, electing GT as the reference tool for the remainder of the book. On such a basis, Chapter 3 is a practical guide to build up a grid.

Chapters 4, 5, and 6 are oriented towards specific applications of GC to EM. In Chapter 4, the attention is focused on the numerical solution of human-antenna interaction problems, using parallel FDTD codes. This is a classical example of applications with a strong HPC demand. The GC solution to the addressed problem demonstrates one of its immediate benefits: the suitability to achieve substantial speed ups with very low costs. This is especially due to the potentially infinite availability of computational nodes to be enrolled in the grid. Most of the time, this implies that the grid operates in a wide area environment, where dedicated connectivity is not necessarily available. This can be critical when referring to parallel applications with an intensive demand on data communication. In such a case, *bandwidth* is a critical factor for success. Consequently, a capillary penetration of broadband connectivity is definitely a key point for future improvement of HPC grid EM applications.

In Chapter 5, the CAE of aperture-array antennas is addressed in a GC framework. The high suitability of GC to fulfill the joint requirements of HPC and cooperative engineering is demonstrated here. The integration of diverse skills and know-how is here exemplified in the interaction among several applications, eventually distributed in a geographical area and heterogeneous both from an architectural and from a system and software point of view. Computational grids, with their amenability to meta applications, represent an ideal environment to promote collaborative design, taking full advantage from Web technologies and integrating them with supercomputing facilities. These appealing tools can be effectively exploited and can achieve an adequate diffusion only when they can easily be adopted by a "standard" user, with no specific skills in the area of GC. This identifies another critical factor for further success—the *simplicity of management and use of grid resources*.

Finally, Chapter 6 focuses on the optimum planning and management of wireless networks. In this case, it has been demonstrated that one more relevant benefit can be added to the two previously described achievements (HPC and cooperative engineering). Namely, we refer to the possibility of a real-time management of data

produced by a heterogeneous and dispersed network of sources (in the discussed example, a network of sensors and antennas with a geographical distribution). The achieved result, of paramount importance for the effective design of current and next generation wireless networks, leads to the definition of a multifolded system, integrating meta applications and distributed data repositories, in an HPC framework. Once again, this relevant result, and the consequent exciting perspective of future developments, goes along with the complexity and multiplicity of the involved technologies. The previously mentioned *simplicity of management and use of grid resources* is even more urgent (resources in this case being data, computers, and application components).

7.2 GC Trends

The two issues on which we focused in the previous sections (bandwidth and simplicity of management and use of grid resources) effectively reflect the current trends in GC research. Indeed, concerning bandwidth, several large research efforts are being performed to extend the availability of broadband connectivity. It is worth mentioning the Teragrid project in the United States [1], working at a very large grid supported by optical-fiber technology. In Europe, one leading project is Delivery of Advanced Network Technology to Europe (DANTE) [2], aiming at delivering GEANT, a broadband network connecting a huge number of leading research sites. In Asia, an outstanding activity is promoted by Asia-Pacific Advanced Network (APAN) [3]. The enumerated efforts, the several other similar projects running in the world, and the recent and constant trends in the evolution of telecommunication technologies encourage a substantial confidence on a more and more successful and fruitful exploitation of GC with all its multiple facets.

As for the simplicity of management and use of grid resources, it seems useful to identify four main directions:

1. Grid services;
2. Automation;
3. Portals;
4. Grid database access and management (GDAM).

Grid services [4, 5] come from the integration of grid concepts and technologies with Web services. Web services [6] represent the natural answer to several questions put forth by the applications proposed in Chapter 5 and 6. Indeed, Web services allow the standardized description of the properties of an application as well as its publication in a Web environment. Once the publication is performed, the published application can be searched for by another application, which automatically can identify and evoke it. In other words, instead of writing a certain part of an application by yourself, you can synthetically describe your specifications and automatically query the Web so that the suitable published application is launched on demand. Consequently, in some cases, the developer's goal is not the *solution* of a problem, it is the rigorous *definition* of the problem.

Automation faces a problem common to grid administrators (i.e., the complex management of grid resources). Grids, for their nature, are intrinsically prone to the drawback of unpredictable modifications of their configuration. Managing such a problem with standard tools for system administrations can be heavy, and the simplification of such procedures is a definite priority. Automation aims at charging grids even with the task of a continuous self reorganization. The interested reader is addressed to [7] for an introductive discussion.

Portals are Web applications allowing a user to connect with a grid, monitor resource status, launch applications, and generally perform all of the grid operations with a point-and-click philosophy. In other words, they are user-friendly interfaces with grid middleware. We suggest the interested reader see [8] for an overview on such a subject.

Finally, GDAM [9] is a forthcoming research area, aiming at integrating data repositories distributed inside a grid so that they are accessed and managed as if they were a unique DBMS. Of course, this is a charming area for the application described in Chapter 6, as well as all data-oriented applications.

Of course, the enumerated four areas are far from representing all of the areas grid research is covering now. The impressive and continuous progress in GC is the best witness of the liveliness of this technology, and one more reason for the EM community to keep a careful eye on the exciting current and future challenges GC offers.

References

[1] http://www.teragrid.org.

[2] http://www.dante.net.

[3] http://www.apan.net.

[4] Foster, I., et al., "Grid Services for Distributed System Integration," *IEEE Computer*, June 2002, pp. 37–45.

[5] http://www.globus.org/ogsa.

[6] Chappel, D., "Examining .NET My Services," *Byte* Spring 2002, pp. 33–40.

[7] Decusatis, C., "Grid Computing: The Next (Really, Really) Big Thing," *Byte*, Spring 2002, pp. 6–13.

[8] http://www.gridcomputing.com.

[9] Paton, N., et al., "Database Access and Integration Services on the Grid," UK e-Science Programme Technical Report Series Number UKeS-2002-03, National e-Science Centre, http://www.cs.man.ac.uk/grid-db/papers/dbtf.pdf.

APPENDIX A

Useful UNIX/Linux Hints

A.1 UNIX/Linux Operating System: An Overview

The UNIX operating system [1, 2] was designed in the 1970s with the goal of providing:

- Simultaneous computer access to a multiplicity of users;
- Easy sharing of information.

The operating system soon became popular among universities and research groups, as it provided a good environment for program development, network transactions, and information sharing. Another remarkable reason for UNIX's immediate success was that it is written in the C high-level language. This makes the operating system highly portable. The other operating systems, in fact, are typically implemented in low-level languages, thus they are strictly tied to the specific platform for which they were developed.

The simplicity and clarity of UNIX programs and the widespread knowledge of the C language tempted many developers to enhance UNIX in their own way. This generated both the enhancement of the operating system with a lot of utilities and services oriented to the end user and the birth of a number of UNIX different *dialects*, running on a variety of platforms, from microprocessors to mainframes. To make order in the myriad of UNIX versions and dialects, the IEEE formed a committee to define standard specifications on operating systems. This work produced a well-known family of standards, called POSIX [3], which still contains the guidelines governing new generation UNIX systems. POSIX documents are not available online but can be purchased in printed form from the IEEE Computer Society [2].

Linux is a completely free reimplementation of UNIX, following most of the POSIX specifications. Its copyright is owned by its designer, Linus Torvalds, and other contributors, and is freely redistributable under the terms of the GNU General Public License (GPL) [4]. GPL classifies it as "free" software, commonly called *freeware* or open source software [5] (i.e., anybody may distribute any modifications, provided that the source for those modifications is distributed as well).

Linux, per se, is only the *kernel* of the operating system (i.e., the part that directly controls hardware, hiding hardware complexity to users). There are several combinations of Linux with sets of utilities and applications to form a complete operating system. Each of these combinations is called a *distribution* of Linux. RedHat Linux [2, 6] is one of the most used Linux distributions.

In the following sections, the most relevant features and commands of the UNIX operating system are overviewed. First, the architecture is sketched in Section A.2. Then, the way UNIX organizes files is overviewed in Section A.3. The multiprogramming environment offered by UNIX systems is well evident when dealing with UNIX processes (i.e., instances of programs in execution), which are treated in Section A.4. A few concepts related to UNIX administration are introduced in Section A.5, and finally the program for user interaction, the so called shell, is overviewed in Section A.6.

A.2 UNIX/Linux: The Architecture

UNIX systems have a layered structure, as depicted in Figure A.1.

In such a schematic description, the hardware is in the center. It includes all of the resources commonly belonging to computing machines (e.g., memory, processor, and devices). The layer surrounding the hardware level represents the UNIX core: the *kernel*. It directly interacts with hardware resources. The kernel provides basic services to users and insulates them and their programs from hardware details. Programs in the outer layers interact with the kernel by invoking a well-defined set of *system calls*. As programs are independent on the underlying hardware, it is easy to move them among UNIX systems running on different hardware platforms, thus guaranteeing what we mean for *portability*. Besides, any user can enhance UNIX systems by adding and distributing programs that provide services oriented to the user community and belonging to the higher level layer.

A.3 The File System

A.3.1 Introduction

As every operating system does, UNIX masks hardware devices for storing data to users. Information stored on disks and commonly distributed in a number of *files* is

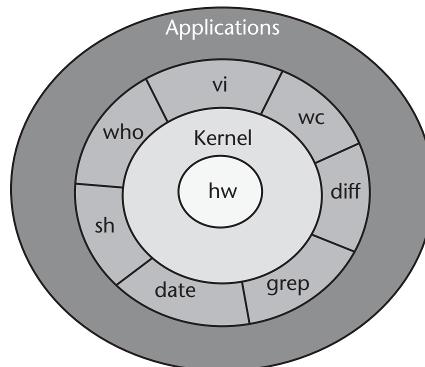

Figure A.1 UNIX layered structure. The hardware at the center includes all of the resources commonly belonging to computing machines (e.g., memory, processor, and devices). UNIX core is the *kernel*, which is the layer directly interacting with hardware resources. The kernel provides basic services to users and insulates them and their programs from hardware details. Programs shown in the outer layers interact with the kernel by invoking a well-defined set of *system calls*.

accessed by interacting with the *file system*. A file system provides a logical view of file data with the scope of hiding the file internal format and hardware operations to end users.

The UNIX file system is characterized by a hierarchical structure (see Figure A.2) with a root node commonly named with the "/" character. Every nonleaf node is a directory of files (i.e., a logical file container), while leaf nodes may be either files or directories. The name of a file is given by the path name that describes how to locate the file in the file system hierarchy together with the name of the file. In Section A.3.2, we recall the most used commands to move inside file systems and to manage files and directories. In Section A.3.3, we explain how to build pathnames. Then we overview the most used system calls for managing files from inside programs (Section A.3.4) and the mechanism used to restrict access on files (Section A.3.5).

A.3.2 File System Relevant Commands

To move inside the file system, the user can type:

cd pathname

where "pathname" is the name of the target directory.

To create a new directory:

mkdir pathname

To remove an empty directory (of course, should the directory contain some files, you must remove them first) type:

rmdir pathname

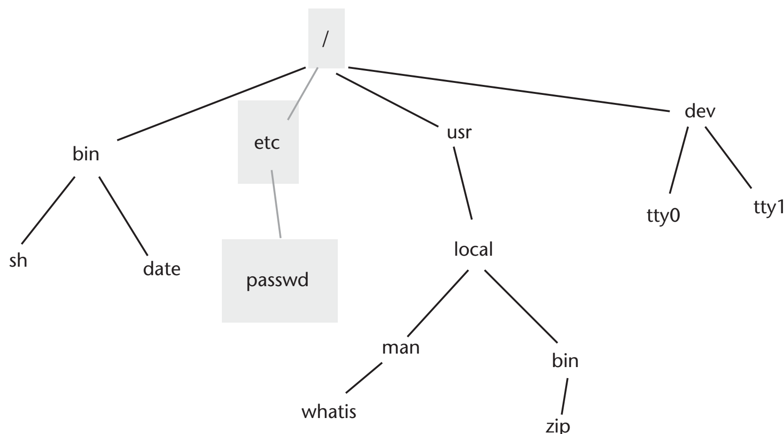

Figure A.2 UNIX file system tree structure. The root node is named with the "/" character. Every nonleaf node is a directory of files, and leaf nodes are either files or directories. The name of a file is given by the path name that describes how to locate the file in the file system hierarchy. For example, the file named "passwd" is located by specifying the path name "/etc/" and concatenating it with the file name, thus obtaining the full file name "/etc/passwd."

To remove a file, type:

rm pathname/file

To move a file, type:

mv pathname1/file1 pathname2/file2

This command moves the file named "file1" located in the directory called "pathname1" into the file "file2" in the directory named "pathname2." The result is the deletion of the file named file1 and its copy moved onto the file named file2. When the pathname of the original file and the pathname of the target file are the same, the command just renames the file.

To copy a file, type:

cp pathname1/file1 pathname2/file2

This command copies the file named "file1" located in the directory called "pathname1" onto the file "file2" in the directory named "pathname2." The result is a couple of identical files: the file named file1 and its copy named file2.

To make a *symbolic link* (i.e., a pointer to the name of a file), type:

ln –s pathname1/file1 pathname2/file2

This command links the file named "file1" located in the directory called "pathname1" with the file "file2" in the directory named "pathname2." The result is a single file stored in the disk accessible from two locations: the pathname named pathname1 and the pathname named pathname2 (where it is called file2).

To compress a file, type:

gzip pathname/myfile

This command produces the file named "myfile.gz," containing the same information as the original file in a reduced space.

To decompress a file, enter:

gzip –d pathname/myfile.gz

or

gunzip pathname/myfile.gz

These commands return the original file and assign it the name "myfile." To have these commands addressing their output on standard output, without generating any new files, the command to enter is:

gzip –dc pathname/myfile.gz

or

gunzip –c pathname/myfile.gz

To aggregate a multiplicity of files in an archive, the *tar* command must be used. For example, if you need to store all of the files contained in the current directory in an archive called “archive.tar,” type:

*tar –cvf archive.tar **

This command produces the file called “archive.tar,” which aggregates the inputted files. To extract files from it, enter:

tar –xvf archive.tar

A.3.3 Pathnames

Pathnames can be *absolute* or *relative*.

A full (absolute) pathname is given by the sequence of the names of the nodes traversed when navigating in the tree from the root up to the leaf representing the targeted file. A full pathname always begins with the “/” character, which identifies the file system root.

For example:

cd /usr/local

asks to move to the directory named “local” and located under the directory named “usr” in the file system hierarchy.

A *relative* path name is a path name obtained considering the subtree rooted at the current directory. For example, suppose that the current directory is */usr/local*, then by typing

cd bin

we move to the */usr/local/bin* folder.

A special meaning is associated to the symbols “.” and “..”—the dot means the current directory. For example, the command:

./a.out

requests the launching of the application named a.out, located in the current directory.

The double dot symbol represents the directory immediately above the current one in the file system tree. For example, if the current directory is “/home/alexa/src,” the command

cd ..

moves to the “/home/alexa/” folder.

A.3.4 System Calls for File Management

The kernel exhibits file data to users as unformatted streams of bytes. Note that in UNIX, systems directories and I/O devices are represented with unformatted streams of data as well, thus providing a uniform way to access data.

A number of system calls allow programmers to access these streams and interpret them as they wish. The *open*, *read*, *write,* and *close* system calls, for example, are the basic system calls to access and modify file data. They resemble the fopen, fread, fwrite, and fclose operations available in C libraries. The *open* system call must be invoked each time a program needs to access a file. It returns an integer, the *file descriptor,* which must be used for subsequent references to the file. The close system call must be invoked to release the resource. For example, a typical code to access a file in read mode may contain the following lines:

```
fd=open("/home/alexa/myfile,""r");
read(fd,data);
close(fd);
```

where fd is the file descriptor.

A.3.5 Permissions

In UNIX, every user is a member of a *group*. Each file in the directory structure is owned by a user and is associated to the group to which the user belongs. Permissions define what users are allowed to do with the file. There are three basic things users might do with a file: read from it, write onto it, and execute it (when executable, of course). Permissions are based on this concept, combined with users and groups definitions.

Permissions are set by using the *chmod* command, with the basic format:

chmod xyz file

Where x, y, and z may each assume an integer value between 0 and 7 and represent each the permissions granted to a different group of users:

- x is for the user that owns the file.
- y is for the group that owns the file (normally the user's group).
- z is for everybody else.

The values x, y, and z can assume are listed in Table A.1, together with their meaning:

For example, the command:

chmod 600 myfile

says that the file named "myfile" has the following access restrictions:

- $x = 6$ means that the user can read and write on it, without being allowed to execute it.

Table A.1 Access Permissions

Value	*Permissions*		
	Read	*Write*	*Execute*
0	N	N	N
1	N	N	Y
2	N	Y	N
3	N	Y	N
4	Y	N	N
5	Y	N	Y
6	Y	Y	N
7	Y	Y	Y

- $y = 0$ means that members of the user group can do nothing with the file.
- $z = 0$ means that other users can do nothing with the file.

In conclusion, "myfile" is not an executable and can be read and written by the user and by nobody else.

A.4 Processes

A *program* is an executable file, while a *process* is an instance of the program in execution. Many processes can execute simultaneously on UNIX systems (the *multiprogramming* or *multitasking* feature), whether they are instances of the same program or of different programs. Processes are active entities with their own life cycle and capability to interact with one another, to create other processes, or to change their behavior, depending on the occurrence of specific events. UNIX contains all of the system calls needed to develop programs giving place to such active processes. The most relevant are *fork* and *exec*. The fork system call allows a process to create a new process, while the exec system call allows a process to execute a program inside its own environment.

A process environment is the ensemble of files, variables, and data that govern the execution of the related program. For example, each process is associated with an *execution directory* and with three files.

The execution directory is the directory where the process is launched and where the files it creates are located. The three files that processes are associated with are the *standard input*, *standard output*, and *standard error*. Processes read from their standard input, write to their standard output, and address their error messages to their standard error. Processes typically associate these three files to hardware devices such as video display and keyboard: they give output on the screen and take input from the keyboard. Users can change the standard input, output, or error by *redirecting* them on another file, as described in Section A.6. The files where standard input, output, and error are redirected are created in the execution directory of the process, if no absolute path is used to name these files.

Special UNIX processes are *daemon* processes. Daemons are processes that are not normally associated with a user, but rather with the operating system, as they support system general functions, such as administration, control of networks, and

line printer spooling. Most daemons are automatically launched at the system boot and run in the *background* (i.e., as later explained in Section A.6, they are not associated with any terminal, and their running is unrelated to user interaction), waiting for a request of service coming from the kernel.

A.5 Administration

Administration consists of making a number of functions for the general welfare of the user community, such as network configuration, disk formatting, and creation of file systems. UNIX administration is performed by a specific user account that has special privileges. This special account is normally called *root* [2] and is recognized by UNIX as having the right to access and manipulate a number of files and programs otherwise off limits to general users.

The administrator can add users to the system by issuing the following command:

adduser username

where username is the login name the administrator associates to the new user.

With this command, each user is given an account (i.e., a working environment where he can access by entering the username and a secret password). By default, when a user logs into a machine, he accesses his own portion of the file system, which is rooted at a directory specifically created for him, called *home directory*.

The administrator can also define *startup services* (i.e., services running at boot time). They may be daemon programs running in the background or one-time-only programs running during the bootstrap to provide functions to the system. Startup services can be defined by updating some initialization scripts that automatically run at startup. A relevant service, installed on several Linux distributions, is the eXtended InterNET services daemon (xinetd) [7], responsible for starting programs that provide Internet services. To communicate to this daemon the services to be started, the xinetd configuration file (/etc/services) has to be updated.

A.6 The Shell

A.6.1 Introduction

The shell is a program managing user interaction with the operating system. Different shell programs are available. They are commonly grouped into two families, each including shells with similar features. The Bourne shell family includes the *sh*, *bash,* and *ksh* shell. The C shell family includes the *tcsh* and the *csh* shell, which is very similar to the C programming language.

The shell is a command language interpreter that executes commands read from the standard input device (keyboard) or from a file. Shells interpret and understand both UNIX commands and internal keywords. The keywords make the shell a true programming language. The shell, in fact, interprets the most typical language structures, such as variable declaration, loops (e.g., *for* and *while*), and conditional

execution (e.g., *if then*). This allows users to write shell *scripts* (i.e., plain text executable files containing shell commands and keywords), which manipulate programs and commands, adding flexibility to the system.

For example, the following shell script:

```
#!/bin/sh
FILES=`find / -name "*.tmp" -print`

for i in $FILES; do
  rm $i
done
```

deletes all files whose name terminates with the ".tmp" string.

The first line (beginning with the "#" symbol) specifies the shell to use. The second line assigns to the local variable named "FILE" the result of the UNIX command:

find / -name ".tmp" –print*

This command searches the entire file system for files having a filename terminating with the ".tmp" string. The standard output of this command (i.e., the list of the files) is assigned to the FILE variable.

The loop (beginning with the "*for*" keyword and terminated by the "*done*" keyword) cycles on all values contained in the FILE variable and executes the command:

rm $i

This command removes a file at each iteration, as the variable named *i* contains a different file name at each loop cycle.

In addition to the common language structures, such as the already mentioned looping and conditional structures, shells offer a number of special keywords for exploiting several interesting features that make the UNIX systems particularly powerful and effective for application developers:

- *Background command execution*. Shells allow asynchronous execution of commands (i.e., when the user types the command, the shell launches the requested job, reads the next command line, and executes it without having to wait for the prior command to terminate).
- *Redirection*. Shells allow the user to change the standard input, output, and error of processes.
- *Pipes*. Piping is a mechanism that allows *the output of one process to be the input to another process.*
- *Environment variable*. Shells allow users to define variables and to export their value to the whole operating system.

In the following sections, these relevant shell features are briefly described.

A.6.2 Background Command Execution

The shell usually executes a command synchronously (i.e., it waits for the command to terminate before reading the next command line, which means that the user has to wait, too, before typing the next command). In other words, the shell gives control to the user until the user enters a command. When a command is entered, the shell takes back control and gives it to the command/process that the user specified. When the process/command completes, control returns to the user.

Commands executed this way are said to execute in *foreground*. The shell also allows asynchronous execution (i.e., control stays with the user, even while the command/process is executing). In this case, commands are said to run in the *background*. This is very useful when launching time-consuming programs: the user can go on working while the program executes.

Running a command in background is simple: the user has to concatenate the "&" symbol to the command name. For example, by typing:

a.out &

the shell executes the program named "a.out" in the background.

By typing the command:

ps

the user is informed about current running processes.

A.6.3 Redirection

The shell allows users to define the file on which to store standard input, output, and error data. As default, they are the terminal. If the user types:

a.out > myfile

the program named "a.out" writes its standard output on the file named "myfile."

If the user types:

a.out < myfile

the program named "a.out" reads its standard input from the file named "myfile."

In the Bourne shell family, if the user types:

a.out 2> myfile

the program named "a.out" writes its standard error on the file named "myfile."

Redirections can be combined:

a.out < inputfile > outputfile 2> errorfile

A.6.4 Pipes

Pipes allow processes to redirect their standard output to a system buffer, called a *pipe*. Other processes can redirect their standard input to come from the pipe. In this

manner, data are passed among processes through the pipe. The symbol used for piping data is "|". For example, by typing:

gzip archive.tar.gz –dc | tar xf –

the standard output of the command gzip is the standard input to the command tar. So if gzip outputs the file named "archive.tar," the command tar extracts files from it. In this manner, by issuing a single-line command, the user decompresses the archive and extracts files from it.

A.6.5 Environment Variables

Shells allow users to define the *environment variables* (i.e., variables whose value is known by the operating system and the execution environment). A number of them (called system variables) are set by the operating system. In Table A.2, you can find a list of the most meaningful.

The full list can be retrieved by issuing the *env* command, which prints all of the system variables with their current value.

To define your own environmental variables, the command to be issued depends on the shell being installed. In Table A.3, the commands for the most used shells are listed.

To access the value of a variable, the "$" character must be used. A common use of environment variables is for the pathname of widely used tools. For example, after having installed a tool in the directory named "/usr/local/tool," the tool can be invoked without entering the full pathname by defining an environment variable. In case of a Bash shell enter:

export $TOOLDIR=/usr/local/tool

From now on, the tool can run by typing:

$TOOLDIR/tool_name

Table A.2 List of Meaningful Environmental Variables

System Variable	*Example*	*Meaning*
HOME	/home/alexa	Home directory
OSTYPE	Linux	Operating System type
PWD	/home/alexa/programs	Current working directory
SHELL	/bin/bash	Shell name
USERNAME	alexa	User name who is currently logged

Table A.3 Commands for the Most Used Shells

Shell	*Command*
C shell	*setenv VARIABLE_NAME value*
Bourne shell	*export VARIABLE_NAME=value*

where *tool_name* is name of the executable.

When the user logs into her account, the system runs a shell script located in her home directory, usually referred as *login, start-up,* or *profile* script. The user can customize her working environment by editing this script, where she can add environment variables, run her own scripts, launch commands, and so on.

References

[1] Bach, M. J., *The Design of the Unix Operating System*, Englewood Cliffs, NJ: Prentice-Hall. 1987.

[2] Nemeth, E., et al., *Unix System Administration Handbook*, Englewood Cliffs, NJ: Prentice Hall PTR, 1999.

[3] Lewine, D., *POSIX Programmer's Guide*, Sebsastopol, CA: O'Reilly & Associates.

[4] ftp://prep.ai.mit.edu/pub/gnu/COPYING.

[5] http://www.opensource.org.

[6] http://www.redhat.com.

[7] http://www.xinetd.org.

APPENDIX B

Foundations of Cryptography and Security

B.1 Introduction

When exchanging data through a network, a number of entities are involved:

- The sender (i.e., the party who sends the message);
- The receiver (i.e., the party to whom the message is addressed);
- The channel (i.e., the medium transporting messages from the sender to the destination).

The Internet is an example of a channel. It is an *open* and *public* channel (i.e., its technology is adherent to well-known international standards, and anybody can use it). Internet openness is one of the main causes of its strength and wide penetration, even though this implies some drawbacks as well. The most critical is *security:* when a message travels through Internet nodes and cables, anybody may intercept it and eventually modify its contents (Figure B.1). A number of freely downloadable tools exist that allow anybody to "sniff" packets traveling in the network and eventually alter their data.

Cryptography, a very old science dating back to Egyptian pharaohs age, provides algorithms and techniques to hide information and to tunnel it straight to the legitimate receiver. Cryptography algorithms are nowadays used to manage the

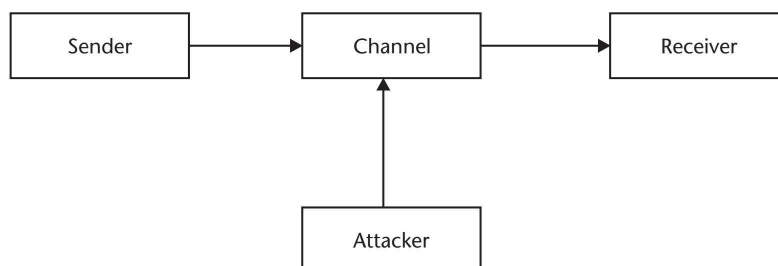

Figure B.1 Information flow in a conventional insecure communication system. There are two legitimate parties: a sender and a receiver. The sender generates a message to be communicated over an insecure channel to the receiver. A third party, the eavesdropper, may read the message or modify its contents while it is traveling through the channel.

most common security requirements: *confidentiality*, *authentication*, and *data integrity*. *Confidentiality* is achieved when sender and receiver are sure that exchanged messages are read by nobody except themselves. *Authentication* deals with verification of authenticity of communicating parties and is performed by apposing a *digital signature* to messages (i.e., binding messages to a specific user). *Data integrity* ensures that data are not modified during their travel from the sender to the destination.

This appendix does not explore all of the basic concepts related to security; rather, it focuses on explaining concepts useful for understanding principles expressed in the previous chapters. In order to have a more exhaustive view of security and cryptography, we refer the reader to [1]. This appendix overviews the most-used technologies to achieve confidentiality (in the following section) and the techniques used for digital signature of documents (Section B.3). As any modern security tool strongly relies on the use of certificates, their meaning is explained in Section B.4.

B.2 Confidentiality and Cryptography

To achieve a confidential exchange of information, *cryptographic* techniques are commonly used. Cryptography [1] is the science of encrypting and decrypting information. *Encryption* means scrambling the text in a complex manner, so that its contents cannot be understood when reading it. Encryption translates the input text, commonly called *plain text,* into an unreadable sequence of characters, called *cipher text*. Only by applying the reverse function to the cipher text (i.e., *decrypting* it) can the original text be obtained.

The first encryption/decryption algorithms were based on the *symmetric key encryption*, which made it virtually impossible to decrypt an encrypted text into plaintext without the use of a secret key (see Figure B.2). The secret key must be known to both the sender and the receiver for encryption and decryption, and must

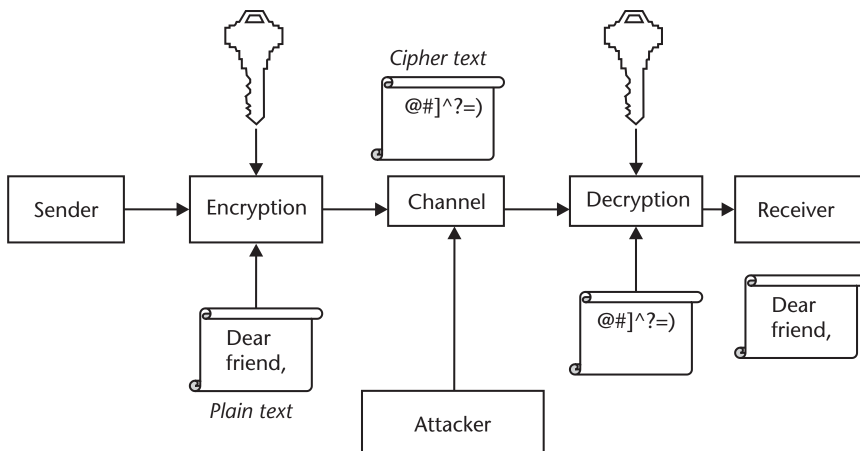

Figure B.2 Symmetric (private) key technique. The sender encrypts the original (plain) text by using a private key. The same key must be used on the receiver end to decrypt the cipher text.

be kept private (this is why this type of algorithms are also called *private key algorithms*).

Symmetric key algorithms are still widely used, even if they have a *key distribution* problem. This problem derives from the fact that both sender and receiver share the same key, which must be kept private. The difficulty is that if communication is to occur, the sender has to tell the receiver what secret key they are going to use without compromising the key's privacy. In order to do this, the sender has to communicate the key to the receiver through a *secure* channel (i.e., a channel that surely cannot be penetrated by anybody). A secure channel may be a courier or direct meeting. The establishment of a secure communication through a secure channel looks like an annoying trouble in the Internet era.

Recently, techniques involving both public and private keys have emerged [2], which solve the key distribution problem. They are the *asymmetric* (or *public*) *key encryption* techniques. Under such schemes, each user owns a couple of related keys, a private one and a public one. Information encrypted by a public key can only be decrypted with the corresponding private key (and vice versa). To start a confidential communication among party A and party B, the following procedure must be followed (see Figure B.3):

1. A asks B to have her public key;
2. B sends A her public key;
3. A encrypts information by using B's public key;
4. B and only B can decrypt cipher text, as B and only B owns the private key associated with the public key used for encryption.

Confidentiality lies in the fact that the decrypting key (i.e. the receiver's private key) is known only by the legitimate receiver.

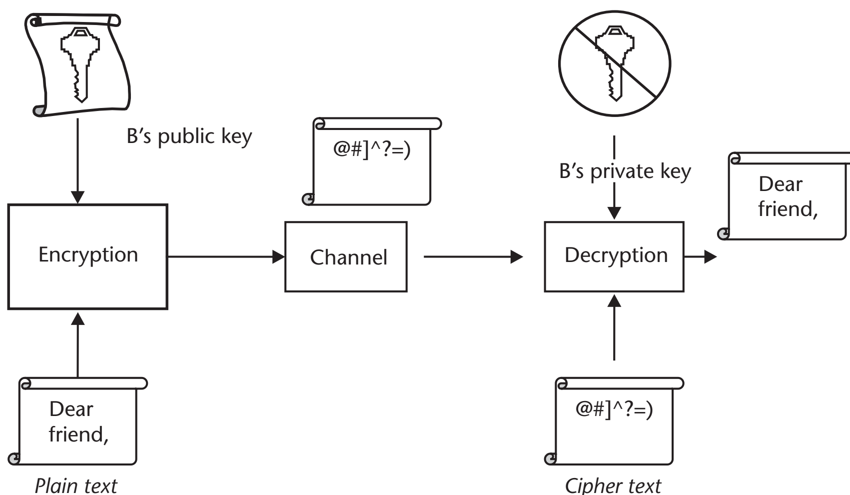

Figure B.3 Asymmetric (public) key technique. Each user owns a couple of keys: a private and a public one. To initiate a private communication, the receiver has to tell the sender his public key. The sender encrypts the text by using the receiver's public key. The receiver and only him can decrypt the text by using his own private key.

This scheme solves the key distribution problem: in fact, the key to be distributed (i.e., the key needed to encrypt text before sending it) is a public one (i.e., everybody can freely access it), while the private key must be held by its owner.

B.3 Digital Signature

Digital signatures substitute handwritten signatures in digital environments. In order to do this, they must follow the following requirements:

- They must be univocally associated to the owner;
- They must be unable to be forged;
- It must be impossible to cut or substitute them;
- They must be verifiable by a third party at each moment.

This is achieved by following the scheme depicted in Figure B.4.

There, the signatory encrypts the document with his or her own private key and joins the plain text of the document with its cipher text. Note that in this manner, the earlier listed requirements are satisfied:

- The signature is univocally associated to the owner, as the private key is such.
- The signature is unable to be forged, as nobody can imitate the private key (if we assume that the owner has taken enough care to keep it safe).
- It is impossible to cut or substitute a signature. In order to substitute a signature, somebody should associate a plain (cipher) text with a different cipher (plain) text with respect to the original one. Well, this is impossible because cryptographic techniques are all designed so that even a single-bit change in a plain (cipher) text implies huge changes in the corresponding cipher (plain) text [1].

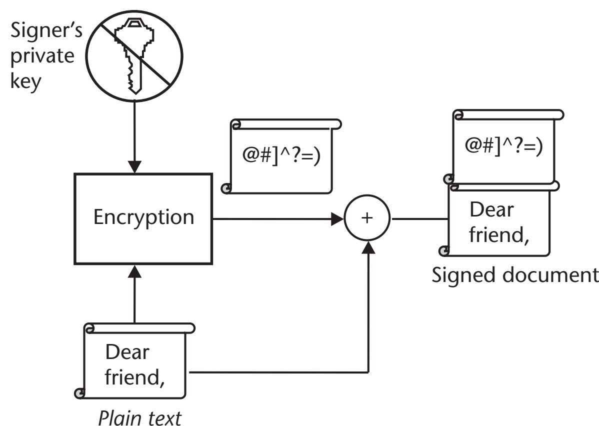

Figure B.4 Digital signature. Public key techniques allow users to perform legally valid digital signatures. To sign a document, it must be encrypted with the signer's private key. The cipher text is then concatenated with the plain text.

- The signature must be verifiable by a third party at each moment. This can be done by following the scheme depicted in Figure B.5. The third party can extract the cipher text and decrypt it with the signatory public key. The result is then compared with the plain text. If the decrypted text matches the plain text, the signature is reputed valid.

B.4 Certificates and Certification Authorities

As explained in Section B.2, public key algorithms solve the key distribution problem. Unfortunately, they are exposed to the “man in the middle” attack.

Suppose that a third party, named C, is able to intercept data traveling in the channel. Suppose also that C is able to substitute B’s public key with his own public key. Imagine now that party A wants to initiate a private communication with party B. The following actions may take place:

1. A asks B to have his public key;
2. B sends A his public key;
3. C substitutes B’s public key with his own public key;
4. A encrypts information by using C’s public key (while believing it is B’s one);
5. C decrypts the cipher text sent by A, by using his own private key.

Therefore, A exchanges private information with C (see Figure B.6), while believing he is talking confidentially with B.

To cope with this risk, the parties involved in the communication elect a trusted party, the CA, with the role of certifying the authenticity of the association between owners and public keys (see Figure B.7).

CAs deliver digital documents, called certificates, containing data about the owner and its public key, such as:

- Owner data;

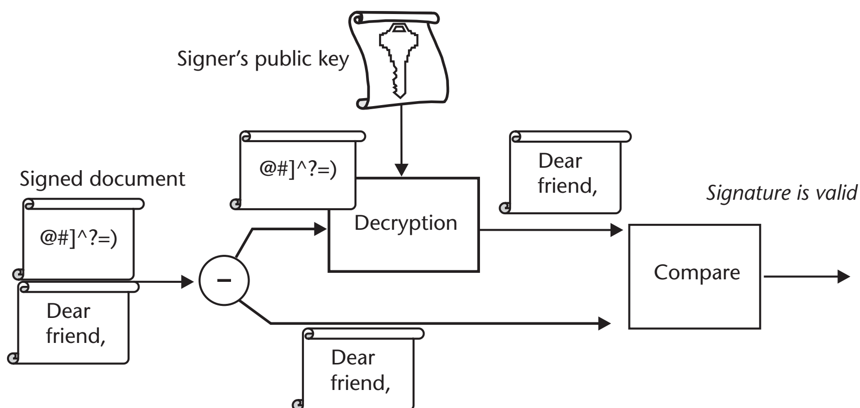

Figure B.5 Digital signature verification. To verify the validity of a signature, the cipher text is extracted and decrypted. The result of the decryption is then compared with the plain text contained in the signature. If they match, signature is reputed valid.

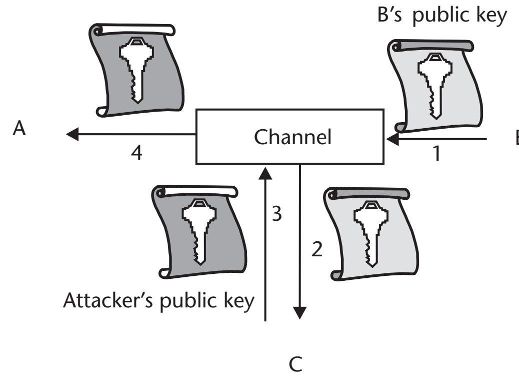

Figure B.6 "Man in the middle" attack. Before sending messages to B, A asks B to send his public key. B sends its own key (arrow 1). An attacker intercepts B's key (arrow 2) and substitutes it with his own (arrow 3). Party A receives the attacker's public key (arrow 4). From now on, A will encrypt data with the attacker's public key (while believing it is B's key). This permits the attacker to read the contents of enciphered text sent by A and eventually to substitute it with other text (encrypted by the attacker with B's public key).

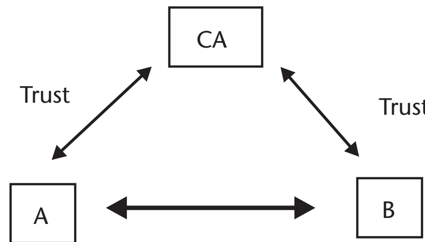

Figure B.7 To cope with the man-in-the-middle attack, a third party is involved. Receiver and sender (A and B in the figure) choose a CA that both trust. The CA will guarantee the validity of associations between public keys and owners by signing documents (certificates) expressing the associa-

- Public key;
- Expiration date of the public key.

A certificate is issued by the CA only after having verified the trustworthy of the owner (i.e., after verifying he is who he claims to be). In order to express their trust, CAs digitally sign each certificate they deliver.

When the sender asks for the receiver's public key, in order to initiate a confidential exchange of information, the receiver responds with his signed certificate. The verification of the CA signature assures the sender that the binding between owner and public key contained in the certificate is true.

A number of internationally recognized CAs exist [3, 4], which deliver certificates after making a number of controls on the authenticity of requesters. In e-business applications, where information is exchanged with unknown people, it is worthwhile to appeal to well-known and worldwide recognized CAs. In environments where the participants are more restricted, a more suitable choice may be to define an autonomous CA that everybody in the community trusts.

References

[1] Menezes, A., P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, Boca Raton, FL: CRC Press, 1996.

[2] Diffie, W., and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, Vol. IT-22, No. 6, November 1976.

[3] http://www.verysign.com.

[4] http://www.thawte.com.

APPENDIX C

Foundations for Electromagnetic Theory

C.1 Maxwell's Equations in the Time Domain

The differential forms of Maxwell's equations in time domain are quite well known to everyone involved in EM research. Nonetheless, it is worth recalling them, as well as their main derivations.

We assume that:

$E = E(r,t)$ is the electric field, expressed in V/m.

$H = H(r,t)$ is the magnetic field, expressed in A/m.

$D = D(r,t)$ is the electric flux density vector, expressed in C/m^2.

$B = B(r,t)$ is the magnetic induction, expressed in Wb/m^2.

$J = J(r,t)$ is the electric current density, expressed in A/m^2.

$\rho = \rho(r,t)$ is the electric charge density, expressed in C/m^3.

$J_m = J_m(r,t)$ is the magnetic current density, expressed in V/m^2.

One of Maxwell's equations is strictly connected with Faraday's law and has the following form:

$$\nabla x\overline{E} = -\frac{\partial \overline{B}}{\partial t} - \bar{J}_m \tag{C.1}$$

The second Maxwell's equation is strictly connected with Ampere's law and has the following form:

$$\nabla x\overline{H} = \frac{\partial \overline{D}}{\partial t} + \bar{J} \tag{C.2}$$

From Gauss's Law, we can derive that:

$$\nabla \cdot \overline{D} = \rho \tag{C.3}$$

with its dual equation:

$$\nabla \cdot \overline{B} = 0 \tag{C.4}$$

It is worth recalling that the magnetic current density is not a physical entity (equivalently, we assume that no magnetic charges can exist).

When dealing with linear, isotropic, and nondispersive (in time and space) materials, the previously mentioned equations are accompanied by the following constitutive equations:

$$\overline{D} = \varepsilon \overline{E} \tag{C.5}$$

$$\overline{B} = \mu \overline{H} \tag{C.6}$$

where ε and μ are, respectively, the electric and magnetic permeability. In vacuum, they assume the following values:

$$\varepsilon_o \approx 1/(36\pi)\cdot 10^{-9} \quad F/m \tag{C.7}$$

$$\mu_o \approx 4\pi \cdot 10^{-7} \quad H/m \tag{C.8}$$

The following constitutive equation holds as well (local Ohm's law):

$$\bar{J} = \sigma \overline{E} \tag{C.9}$$

where σ is the electric conductivity (S/m). Joining Maxwell's equations with the constitutive equations, the following system of six scalar equations is derived, referred to a 3D rectangular coordinate system:

$$\begin{aligned}
\frac{\partial H_x}{\partial t} &= \frac{1}{\mu}\left(\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y}\right) \\
\frac{\partial H_y}{\partial t} &= \frac{1}{\mu}\left(\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z}\right) \\
\frac{\partial H_z}{\partial t} &= \frac{1}{\mu}\left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x}\right) \\
\frac{\partial E_x}{\partial t} &= \frac{1}{\varepsilon}\left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma E_x\right) \\
\frac{\partial E_y}{\partial t} &= \frac{1}{\varepsilon}\left(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} - \sigma E_y\right) \\
\frac{\partial E_z}{\partial t} &= \frac{1}{\varepsilon}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma E_z\right)
\end{aligned} \tag{C.10}$$

The discretization with a time step Δt, and a space step Δx, Δy, and Δz along the x, y, and z axis, respectively, leads to equations such as the one reported in (4.1).

C.2 Helmholtz and Dispersion Equations

When impressed electric currents $\boldsymbol{J}_i$ and magnetic currents $\boldsymbol{J}_{mi}$ must be taken into account, the introduction of a vector magnetic potential $\boldsymbol{A}$ so that $\boldsymbol{H} = \nabla x \boldsymbol{A}$ and a

vector electric potential $\boldsymbol{F}$ so that $\boldsymbol{E} = \nabla x \boldsymbol{F}$ are useful, so that the introduction of suitable gauge terms leads to the inhomogeneous Helmholtz equation:

$$\nabla^2 \overline{\boldsymbol{A}} - \mu\varepsilon \frac{\partial^2 \overline{\boldsymbol{A}}}{\partial t^2} = -\bar{\boldsymbol{J}}_i \tag{C.11}$$

or its dual form.

This equation allows the solution of radiation problems, as well as the introduction of the concept of plane wave and its phase velocity, with respect to a certain direction $\boldsymbol{a}$, indicated with v_a. Recalling that the velocity of light $c = \frac{1}{\sqrt{\mu_o \varepsilon_o}} \approx 3 \cdot 10^8\ m/s$ in vacuum, along the direction of propagation β, we have:

$$v_\beta = \frac{\omega}{\beta} = c \tag{C.12}$$

where $\omega=2\pi f$, and f is the working frequency. The homogeneous Helmholtz equation, derived from (C.11) by assuming that no impressed currents exist, and the method of variable separation lead to the introduction of three variables k_x, k_y, and k_z, so that the following dispersion relation must hold for a plane wave in a homogeneous lossless medium:

$$k_x^2 + k_y^2 + k_z^2 = \frac{\omega^2}{c^2} \tag{C.13}$$

Equation (C.13) can easily be customized for the discretized three dimensional case suited to FDTD modeling. Indeed, after introducing a time step Δt, and a space step Δx, Δy, and Δz along the x, y, and z axis, respectively, for a plane wave propagating inside the FDTD mesh, the following discrete equation holds:

$$\begin{aligned} \frac{1}{(c\Delta t)^2} \sin^2\left(\frac{\omega \Delta t}{2}\right) &= \frac{1}{\Delta x^2} \sin^2\left(\frac{k_x \Delta x}{2}\right) \\ &+ \frac{1}{\Delta y^2} \sin^2\left(\frac{k_y \Delta y}{2}\right) + \frac{1}{\Delta z^2} \sin^2\left(\frac{k_z \Delta z}{2}\right) \end{aligned} \tag{C.14}$$

Equation (C.14) degenerates into (C.13) when space and time steps go to zero.

C.3 TE and TM Modes

Waves inside a guiding structure can be studied by adding suitable boundary conditions to Helmholtz equation. In the case of rectangular waveguides, a generic field can be intended as a linear combination of waveguide modes, such as TE and TM modes. Assuming that the x axis of a rectangular system is along the main edge of the waveguide section (with dimensions $a \times b$), for TE modes we have:

$$\bar{e}_{mn} = N_C \left(\bar{x}_o \frac{n\pi}{b} \cos\frac{m\pi x}{a} \sin\frac{n\pi y}{b} - \bar{y}_o \frac{m\pi}{a} \sin\frac{m\pi x}{a} \cos\frac{n\pi y}{b} \right)$$
$$\bar{h}_{mn} = N_C \left(\bar{x}_o \frac{m\pi}{a} \sin\frac{m\pi x}{a} \cos\frac{n\pi y}{b} + \bar{y}_o \frac{n\pi}{b} \cos\frac{m\pi x}{a} \sin\frac{n\pi y}{b} \right) \quad \text{(C.15)}$$

with N_C a normalizing constant and with $m, n \geq 0$ (provided that they are not contemporarily equal to zero). For TM modes we have:

$$\bar{e}_{mn} = N_C \left(\bar{x}_o \frac{m\pi}{a} \cos\frac{m\pi x}{a} \sin\frac{n\pi y}{b} + \bar{y}_o \frac{n\pi}{b} \sin\frac{m\pi x}{a} \cos\frac{n\pi y}{b} \right)$$
$$\bar{h}_{mn} = N_C \left(\bar{x}_o \frac{n\pi}{b} \sin\frac{m\pi x}{a} \cos\frac{n\pi y}{b} - \bar{y}_o \frac{m\pi}{a} \cos\frac{m\pi x}{a} \sin\frac{n\pi y}{b} \right) \quad \text{(C.16)}$$

with $m, n \geq 1$.

For TE modes, the axial component is:

$$h_{zmn} = N_C \left(\frac{k_{cmn}^2}{\gamma} \cos\frac{m\pi x}{a} \cos\frac{n\pi y}{b} \right) \quad \text{(C.17)}$$

and for TM modes we have:

$$e_{zmn} = N_C \left(\frac{k_{cmn}^2}{\gamma} \sin\frac{m\pi x}{a} \sin\frac{n\pi y}{b} \right) \quad \text{(C.18)}$$

In (C.17) and (C.18):

$$k_{cmn}^2 = \left(\frac{m\pi}{a}\right)^2 + \left(\frac{n\pi}{b}\right)^2 \quad \text{(C.19)}$$

so that the constant γ_p in (5.1) and (5.2) can be expressed as:

$$\gamma_p = \sqrt{k^2 - k_{cmn}^2} \quad \text{(C.20)}$$

C.4 Fourier Representation of Green's Functions

As is quite well known, the solution of inhomogeneous Helmholtz equations takes substantial advantages from its representation through Fourier integrals. Considering a scalar problem:

$$\nabla^2 \Phi + k^2 \Phi = -w \quad \text{(C.21)}$$

with $\Phi = \Phi(x, y, z)$ and $w = w(x, y, z)$ complex functions, and k complex constant, using Fourier representation we have:

$$\Phi(x,y,z)=\frac{1}{(2\pi)^{\frac{3}{2}}}\int\int\int_{-\infty}^{+\infty}\tilde{\Phi}(\xi,\psi,\zeta)e^{j(\xi x+\psi y+\zeta z)}d\xi d\psi d\zeta \tag{C.22}$$

where $\tilde{\Phi}$ is the three-dimensional Fourier transformation of Φ. In several cases (as, for instance, when studying an aperture in an infinite metal flange), it is a useful 2D transformation. This way, the Green's function for the half space can be written as:

$$G(x-x',y-y') = \int\int_{-\infty}^{+\infty}\frac{df_x df_y}{\sqrt{(f_x)^2+(f_y)^2-\left(\frac{1}{\lambda}\right)^2}}e^{-j2\pi f_x(x-x')-j2\pi f_y(y-y')} \tag{C.23}$$

where $f_x = \frac{1}{\lambda}\sin\theta\cos\varphi$, and $f_y = \frac{1}{\lambda}\sin\theta\sin\varphi$, referring to the spherical coordinate system, as in Figure C.1.

The use of such transformations is a powerful approach, adopted when studying diffraction from apertures and highly attractive when studying the problem in the vicinity of apertures and flanges. In such cases, the formulation leads to the well-known *plane-wave representation*, where the radiated field is considered the summation of elementary contributions. Such contributions are uniform plane waves in the visible region (see Figure C.2) or evanescent plane waves elsewhere.

This is expressed by the following formula:

$$\bar{E}(\bar{r})=\int_{-\infty}^{+\infty}\int\bar{e}(f_x,f_y)e^{-jk(f_x,f_y)\cdot r}df_x df_y \tag{C.24}$$

where the elementary contributions $\boldsymbol{e}e^{-jk\cdot r}df_x df_y$ apparently assume the form of plane waves. Uniform plane waves propagate in all directions in the half-plane $z > 0$. Evanescent waves propagate in all directions parallel to the aperture plane. They must be carefully taken into account in the proximity of the aperture, while they can be omitted far away from the aperture.

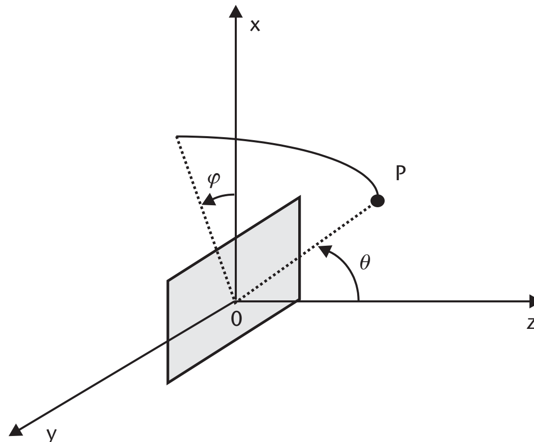

Figure C.1 The spherical coordinate system for the case of aperture antennas.

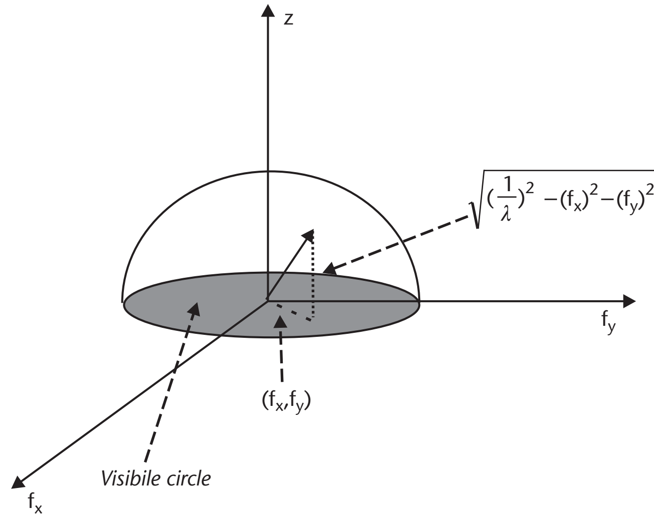

Figure C.2 The visible and evanescent regions.

C.5 The Far-Field Approximation

As concluded in the previous section, in several cases the behavior of EM sources is rather different, depending on the observation distance. The study in the proximity of an antenna is typically more complex, with respect to the analysis at a large distance. This is basically due to the importance that the antenna dimension assumes when reducing the observation distance. Indeed, referring to Figure C.3, this compels us to take into account the different phase of the contributions arriving in P from different regions (identified by $\boldsymbol{r}'$) of the antenna (represented by the gray area).

Referring to [1] for an analytical demonstration, it can be concluded that for distances from the antenna greater than $D_{FF} = \frac{2D^2}{\lambda}$, where D is the leading dimension of the antenna, some relevant simplifications can be assumed, namely:

1. The antenna can be assumed to be a point source.

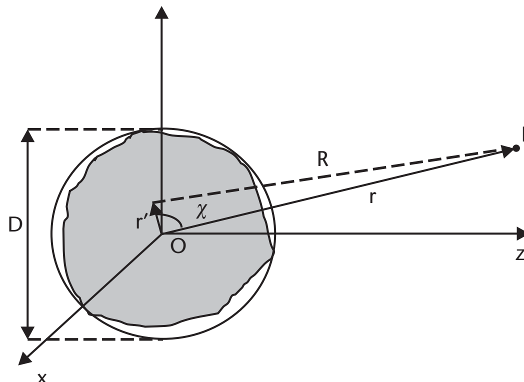

Figure C.3 The importance of the observation distance with respect to the antenna dimensions.

2. The radiated field is a locally TEM wave (i.e., a TEM wave in a small volume around the observation point).
3. The radiation vector depends on the direction of the observation point and does not depend on its distance from the source.

D_{FF} represents a threshold distance. Shorter distances fall inside the so-called Fresnel (or near-field) zone; larger distances fall inside the Fraunhofer (or radiation, or far-field) zone. Near-field problems typically deserve accurate and complex solutions, mostly based on full-wave numerical methods.

Reference

[1] Stutzman W. L., and G. A. Thiele, *Antenna Theory and Design*, New York: John Wiley and Sons, 1981.

APPENDIX D

List of Useful Web Sites

Access Grid Project http://www.accessgrid.org

ASC Portal http://www.acsportal.org

Asia Pacific Grid http://www.apgrid.org

Cactus Code http://www.cactuscode.org

Condor http://www.cs.wisc.edu/condor

DataGrid Project http://www.eu-datagrid.org

Global Grid Forum (GGF) http://www.gridforum.org

Globus http://www.globus.org

Gridbus http://www.gridbus.org

Grid Computing Info Center http://www.gridcomputing.com

Grid Today http://www.gridtoday.com
http://www.csse.monash.edu.au
http://hotpage.npaci.edu

Gridlab http://www.gridlab.org

Grid Physics Network (GriPhyN) http://griphyn.org

IBM Grid http://www.ibm.com/grid

JXTA http://www.jxta.org

Legion http://legion.virginia.edu

The Message Passing Interface Standard (MPI) http://www-unix.mcs.anl.gov/mpi

NCSA http://www.ncsa.uiuc.edu

NSF http://www.nsf.gov

OMG http://www.omg.org

Peer-to-Peer Working Group http://www.peer-to-peerwg.org

SRB http://www.npaci.edu/dice/srb

Teragrid http://www.teragrid.org

W3C http://www.w3.org

WebFlow http://www.npac.syr.edu/users/haupt/WebFlow

Glossary

Abstraction Feature of the object-oriented programming model, according to which entities (objects) having common properties can be grouped.

Abstract device interface (ADI) Code that must be implemented to specialize MPICH for a specific computing platform.

Arithmetic and logic unit (ALU) According to the Von Neumann model, the ALU is a component of the CPU and is responsible for executing the instructions.

Application programming interface (API) APIs facilitate the development of programs using functionalities embedded in software or hardware tools. A tool contains an API when it defines a number of function calls (interfaces) to access its own facilities.

Asymmetric key A separate but integrated key pair, comprised of one public key and one private key. Each key is one way, meaning that a key used to encrypt information cannot be used to decrypt the same data.

Asynchronous Asynchronous tasks execute independently of each other, and their timing is not synchronized. For example, when you start up three asynchronous tasks, even when each does about the same amount of work, you can't predict in which order they will finish.

Authentication Verification of authenticity of communicating parties.

Authorization The procedure for granting access to resources.

Autonomic computing Self-managing computing model named after the human body's autonomic nervous system. An autonomic computing system controls the functioning of computer applications and systems without input from the user, in the same way that the nervous system regulates the body system without conscious input from the individual.

Bandwidth Bandwidth is the total available bit rate of a digital network channel. Bandwidth is determined by the speed of the network, which is determined by its technology, but it is also affected by the overhead of the control data added by the communications protocol.

Barrier Process synchronization mechanism used in collective communications. Processes calling a barrier function block until all the members of the same group have called it.

Beowulf cluster A cluster of Linux based personal computers using commodity hardware and open-source software.

Browser An application program that provides a way to look at and interact with all of the information on the World Wide Web.

Bundle Collection of packages that can be installed together by using a packaging toolkit.

Certification authority (CA) A trusted third party who creates certificates to bind an entity to its public key.

Code division multiple access (CDMA) A spread-spectrum communication technique. Spread-spectrum techniques are based on the signal dispersion over a wider band to reduce sniffing and disturbs. CDMA is a multiplexing technique: several users can adopt the same bandwidth with minimum reciprocal interference. Users are associated to orthogonal codes (Baum-Walsh codes).

Certificate An electronic document attached to a public key by a trusted third party (CA), which provides proof that the public key belongs to a legitimate owner and has not been compromised.

Cipher text Text converted into an unreadable format through the use of an encryption algorithm.

Class Group of objects having common properties and behavior.

Client The requesting program or user in a client/server relationship.

Cluster Group of machines that are networked together and used as a single system.

Commodity Grid (CoG) Set of facilities employed in the Globus Toolkit for integrating Globus with the most common commodity technologies.

Commodity technologies *See* enabling technologies.

Component Object located on a node of a heterogeneous network, able to interoperate with other components, located anywhere on the network, as a unified whole.

Computational speed up *See* speed-up ratio.

Condor Tool supporting high-throughput computing on collections of distributed computing resources.

Confidentiality A service that assures that exchanged messages are known only to the communicating parties.

Common Object Request Broker Architecture (CORBA) Object-oriented distributed computing specification developed by OMG, based on the development of object request brokers able to mediate between clients and server components.

Control processing unit (CPU) According to the Von Neumann model, the CPU is the component of computer machines responsible for fetching instructions from memory and executing them sequentially.

Cryptography The science of creating messages that can be read only by a designated receiver.

Control unit (CU) According to the Von Neumann model, the CU is a component of the central processing unit and is responsible for decoding instructions fetched from memory.

Daemon A program that runs continuously and exists for the purpose of handling periodic service requests that a computer system expects to receive. The daemon program forwards the requests to other programs (or processes) as appropriate.

Data decomposition The division of a global data set into smaller subdomains, typically for distribution over some form of parallel computer.

Distributed computing environment (DCE) Object-oriented distributed computing specification developed by OSF, based on traditional remote procedure calls for the communication between client and server.

DCOM Platform for object-oriented distributed computing developed by Microsoft.

Decryption A method of unscrambling encrypted information so that it becomes legible again.

Delegation The process of accepting some entity acting on behalf of someone else.

Directory Directories are used to store and retrieve information. Thus, directories are similar to databases. Special characteristics of directories include the following: directories are designed for reading more than for writing, directories offer a static view of the data, and updates in directories are simple without transactions.

Directory service A directory service provides a directory that can be accessed via a network protocol. An example of a directory service is domain name system, which resolves the names of computers to appropriate addresses.

Distributed memory system In a distributed-memory system, each processor node has immediate access only to its local memory, so if a processor needs data from another node's memory, it must issue special instructions to fetch these items from that node over the interconnecting network.

Directory information tree (DIT) Hierarchical tree structure used by LDAP containing a hierarchical view of data that makes it amenable for an easy and fast tree-based search system for the data.

Data management (DM) Pillar of the Globus Toolkit dealing with access and management of data in a grid (for instance, it provides a more robust and high-performance file transfer protocol, customized to grid needs).

Distinguished name (DN) Unambiguous name to identify entities in a distributed environment.

Domain name system (DNS) Directory service used to locate computers in the Internet.

Extended access control list (EACL) List defining policies for security.

Efficiency Ratio between the computational speed up and the number of processors operating in parallel.

Enabling technology Basic technology that enables the specification of a higher level technology.

Encapsulation Feature of object-oriented programming paradigm according to which objects hide data and expose a well-defined interface allowing users to operate on hidden data.

Encryption A method of scrambling information to render it unreadable to anyone except the intended recipient

Floating-point operations per second (FLOPS) Measure of computer performance.

Freeware Software freely redistributable under the terms of the GNU General Public License.

Flynn's taxonomy A classification system for computer architectures.

Fully qualified domain name (FQDN) The combination of hostname and domain name of a computer.

File transfer protocol (ftp) Standard protocol for transferring files via a network.

Globus Access to Secondary Storage (GASS) Service of Globus implementing a variety of automatic and programmer-managed data-movement and data-access strategies, enabling programs to read and write remote data.

Gatekeeper User interface to the Globus component called GRAM. When a job is submitted, the gatekeeper authenticates the request and creates a job manager to handle it.

Grid index information system (GIIS) Aggregate directory service of Globus that is able to gather data from multiple nodes of a grid and to respond to aggregate queries.

Globus Toolkit Open-source middleware set of grid services addressing fundamental issues, such as security, information discovery, resource management, data management, and communication.

Grid middleware (GM) Layer of software mediating between resource and high-level application to enable grid computing.

General Public License (GPL) Guidelines to distribute freeware software, according to which freeware software can be modified and redistributed provided that the source for those modifications are distributed as well.

Globus Packaging Toolkit (GPT) Package for a totally automatic installation of the Globus Toolkit.

Grid architecture for computational economy (GRACE) Middleware infrastructure coexisting with grid middleware systems to support computational economy.

Globus Resource Allocation Manager (GRAM) Component of the resource management pillar of the Globus Toolkit responsible for sets of resources operating under the same allocation policy.

GridFTP Extended version of the file transfer protocol (ftp), included in the Globus Toolkit, which adds a series of features to ftp that customizes it to grid environments.

Grid resource information service (GRIS) Globus component responsible for providing information about the status of resources available in each grid node.

Globus Security Infrastructure (GSI) Globus Toolkit component ensuring fundamental security services, such as authentication, confidentiality, and integrity.

Global System for Mobile Communications (GSM) A second-generation cellular system introduced to standardize wireless communications throughout Europe in 1990. The mobile entity communicates with the base station of the cell to which it belongs. The base station connects the mobile with the other mobiles or with the wired network, thanks to a switching center. Each mobile is equipped with a SIM card, with the user id number and all of the information about the user.

GT *See* Globus Toolkit.

Hash One-way hash function has the following features: it is easy to generate an output from it, and it is very difficult to generate the original input from the produced output. Examples would be a sine to inverse sine function or square to square root.

Hypertext transfer protocol (http) Standard protocol for transferring files via the Web.

HTTP over SSL (HTTPS) HTTP protocol enriched with security services provided by secure sockets layer.

Interface definition language (IDL) Language to implement CORBA-compliant components.

Inheritance Form of polymorphism that allows users to define groups of classes specializing operations and attributes owned by other classes.

Interface Function call with a very rigorous and permanent specification, defined to hide implementation details of objects, devices, and tools.

Internet Engineer Task Force (IETF) International community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet.

Integrity Service of assuring and verifying that a message arrives at a destination without any corruption.

Intranet Network of computers belonging to a single organization and adopting Internet standards and protocols.

Information services (IS) Pillar of the Globus Toolkit, responsible for collecting and returning information about the structure and state of resources (e.g., their current load and usage policy).

Java Interpreted object-oriented language widely deployed in the Internet.

JavaBeans Object-oriented distributed computing framework based on the Java language.

Jini Object-oriented distributed computing framework based on the Java language and services.

Job manager Process created by the GT gatekeeper to handle jobs running on server machines.

Kernel Core of the UNIX operating system. It hides hardware complexity from users.

Key A digital code used to encrypt, sign, decrypt, and verify messages and files.

Local area network (LAN) Network connecting computers belonging to a single organization and not distant from each other.

Latency The time taken to start up an operation. Typically message latency is the time delay incurred between one processor starting a message send operation and the recipient processor completing the receive operation. Startup latency is the constant communication overhead incurred in sending a zero-length message.

Legion Object-oriented middleware framework for grids.

Lightweight directory access protocol (LDAP) Software protocol for enabling anyone to locate organizations, individuals, and other resources, such as files and devices, in a network.

Library A collection of precompiled routines that can be linked to a program.

Linux Free reimplementation of the POSIX UNIX specification.

Load balance A measure of how evenly work is distributed among a set of parallel processors. Parallel programs are most efficient when the load is perfectly balanced.

Makefile UNIX utility for compiling applications.

Master-worker Programming paradigm where a root process (master) is responsible for distributing problem data among the remaining processes (workers) and collecting results at the end of the executions.

Metacomputing directory service (MDS) Core of the IS pillar of Globus. MDS has a distributed architecture. It is basically composed of grid resource information services (GRIS) and grid index information systems (GIIS). Each GRIS provides information about the status of resources available in each node. Each GIIS gathers data from multiple GRIS resources.

Message passing Programming paradigm for developing parallel applications. It is the explicit exchange of messages between processes.

Middleware Software acting as intermediate between higher and lower layers in a hierarchical architecture.

Multiple instruction multiple data (MIMD) Parallel architecture containing a number of CPUs interconnected by a high-speed network. The different CPUs execute in parallel different instructions and operate on different data.

Mobile agent Program with the ability to transfer itself from host to host within a network and to interact with other agents in order to perform its task.

Message passing interface (MPI) A standard application programming interface that can be used to create parallel applications.

MPICH Public domain implementation of MPI.

MPICH-G2 Grid-enabled implementation of MPICH.

Massively parallel processor (MPP) Computers containing hundreds or thousands of processors interconnected by a fast local interconnection network.

Multithreaded application Application in which a number of tasks are carried out in parallel by simultaneously running threads.

Mutual authentication The process of authenticating both parties involved in a communication.

Nimrod-G Resource management and scheduling system built on Globus services and freely available on the Internet.

Network time protocol (NTP) Internet standard to synchronize dispersed machines with a common reference clock.

OpenLDAP Freeware LDAP library.

OpenMP A standard API for distributing work across threads of a shared memory computer.

OpenSSL Freeware SSL library.

Object request broker (ORB) Entity responsible for locating components in a distributed object-oriented environment.

Peer to peer Network of computers where each machine can act both as client and as server.

Plain text Text in a human-readable form or bits in a machine-readable form.

Polymorphism Feature of the object-oriented programming model, according to which classes can overlap and intersect (i.e., they can include a common set of operations, eventually assuming different meanings depending on the class to which they are applied).

Portal A term for a World Wide Web site that is or proposes to be a major starting site for users when they get connected to the Web or that users tend to visit as an anchor site.

Portable Operating Systems for Computing Environments (POSIX) Standard containing the guidelines that govern the new generation of operating systems.

Private key Key used to encrypt or decrypt text, associated with a public key.

Process Instance of a program in execution.

Protocol Set of rules that end points use when they communicate.

Proxy Entity acting on behalf of someone else.

Pthreads Threads programming interfaces compliant with the standard specifications included in the Portable Operating System for Computing Environments family of standards.

Public key Key used to encrypt or decrypt text, associated with a private key.

Parallel virtual machine (PVM) A subroutine library from Oak Ridge National Laboratory. PVM includes libraries of subroutines callable from C and Fortran programs, plus system support processes for distributed memory parallelism. PVM's goal is to allow the user to create a parallel virtual machine from any heterogeneous collection of machines and networks.

Resource management (RM) Pillar of the Globus Toolkit, responsible for scheduling and allocating resources specifying, for example, resource requirements and the operations to be performed, such as process creation or data access.

Root account UNIX account normally used to perform administration tasks.

Resource specification language (RSL) Language interpreted by Globus components, having a simple syntax that allows specifying job requests and their characteristics.

Scheduler A program that controls which batch job runs next, when resources are available.

Shared memory architecture MIMD architecture where the processors can address a global, shared memory.

Shell Command interpreter program provided for user interaction with UNIX systems.

Servlet Java pieces of code that cooperate with Java-compliant Web servers to provide services to Web clients. A Java servlet can interface Web servers with databases and other back-end services and elaborate data to give back the results to the Web.

Simple instruction multiple data (SIMD) Parallel architecture where a single CU controls a number of ALUs. ALUs execute in parallel the same instruction on different local data.

Single sign on The procedure of authentication via a single insertion of a secret password.

Speed-up ratio Speed gain obtained from the operation of N processors in parallel.

Single program multiple data (SPMD) Parallel programming paradigm, where all tasks execute the same program but on different sets of data. All the nodes receive identical copies of the program to be executed.

Secure sockets layer (SSL) Protocol developed by Netscape to provide security over the Internet. Supports client and server authentication.

Standard error File where a process writes its error messages; by default it is the screen.

Standard input File where a process reads its input; by default it is the keyboard.

Standard output File where a process writes its output; by default it is the screen.

Symmetric key algorithm Algorithm where the encryption and decryption key are the same.

System call Well-defined interface allowing users to interact with the UNIX kernel.

Task Unit of computation, analogous to a UNIX process.

Transfer control protocol/Internet protocol (TCP/IP) The basic communication language or protocol of the Internet.

Time division multiple access (TDMA) A multiplexing technique based on time division. Each user is assigned a certain time slot. The information about the time slot assigned to a single user is crucial both when transmitting and when receiving.

Thread Stream of instructions that can be scheduled to run as if it were a process with an autonomous identity with respect to the program of which it is part of.

Thread safe Functions are said to be thread safe when they can be safely called by multiple threads—data are not corrupted when these functions are concurrently invoked.

Topology Describes the way nodes are interconnected in a parallel architecture.

Universal Mobile Telecommunication System (UMTS) A standard for the third generation of mobile systems. It is based on two standards. The former is the wideband CDMA (W-CDMA, see CDMA). The latter is called TD-CDMA, and is a combination of W-CDMA and TDMA, see TDMA). The basic goal of the third generation of mobile systems (a digital technology) is the full support of multimedia Web services in a mobile context.

UNIX Multiprogramming and multiuser operating system written in the C language.

Uniform resource locator (URL) The unique address for a file that is accessible on the Internet.

Von Neumann model Model describing the functioning of computer machines where a central processing unit sequentially processes instructions stored in memory.

Vector processor A powerful computer processor designed to perform arithmetic to long vectors rather than single numbers.

Wide area network (WAN) Network connecting geographically dispersed computers.

Webflow Web-based visual tool for the development of grid-enabled applications.

Web server Program that, using the World Wide Web's hypertext transfer protocol, serves the files that form Web pages to Web users.

Web services Services (usually including some combination of programming and data) that are made available from Web servers for Web users or other Web-connected programs.

Extended Internet services daemon (Xinetd) Daemon included in a number of Linux distributions that is responsible for starting programs that provide Internet services.

List of Acronyms

ABC Absorbing boundary conditions

ADI Abstract device interface

AFS Analysis of the feeding system

ALU Arithmetic logical unit

AMC Analysis of mutual coupling

APAN Asia-Pacific Advanced Network

API Application programming interface

AS Atomic subtask

BS Base station

CA Certification authority

CAD Computer-aided design

CAE Computer-aided engineering

CDMA Code division multiple access

CE Cooperative engineering

CEM Computational electromagnetics

CGI Common gateway interface

CORBA Common Object Request Broker Architecture

CPU Central processing unit

CTP Consumer TP

CU Control unit

DAG Direct acyclic graph

DANTE Delivery of Advanced Network Technology to Europe

DBMS Database management system

DCE Distributed Computing Environment

DCOM Distributed Component Object Model

DIT Directory information tree

DM Data management

DN Distinguished name

DNS Domain name system

DPSS Distributed Parallel Storage System

DR Data register

DTF Distributed Terascale Facility

DUROC Dynamically Updated Request Online Coallocator

EACL Extended access control list

EM Electromagnetics

ENC encapsulator

ERP Evaluation of the radiation pattern

ESM Evaluation of the scattering matrix

FDTD finite difference time domain

FLOPS Floating-point operations per second

FQDN Fully qualified domain name

ftp File transfer protocol

GASS Globus Access to Secondary Storage

GC Grid computing

GDAM Grid database access and management

GGF Global Grid Forum

GIIS Grid index information system

GIS Geographic Information System

GM Grid middleware

GPL General Public License

GPT Globus Packaging Toolkit

GRACE Grid architecture for computational economy

GRAM Globus Resource Allocation Manager

GRIS Grid resource information service

GSI Globus Security Infrastructure

GSM Global System for Mobile Communications

GT Globus Toolkit

HBM Heartbeat Monitor

HPC High-performance computing

HPS High-performance switch

HPSS High Performance Storage System

HTC High-throughput computing

HTML Hypertext markup language

HTTP Hypertext transfer protocol

HTTPS HTTP over SSL

IDL Interface definition language

IEEE Institute of Electrical and Electronic Engineers

IETF Internet Engineer Task Force

IM Information management

INFN National Institute for Nuclear Physics

IR Instruction register

IS Information services

ISNOP Integrated system for network optimum planning

IT Information technology

I/O Input/output

JMA Java mobile agents

LAN Local area network

LDAP Lightweight directory access protocol

LOS Line of sight

LSF Load Sharing Facility

LW-GE Lewin/Gegenbauer

LW-WG Lewin/waveguide

MB Main block

MDS Metacomputing directory service

MIMD Multiple instruction multiple data

MM Mode matching

MPI Message passing interface

MPIF Message Passing Interface Forum

MPMD Multiple program multiple data

MPP Massively parallel processor

MW Microwave

NCSA National Center for Supercomputing Applications

NLOS Nonline of sight

NTP Network Time Protocol

OH Okumura-Hata

OO Object oriented

OPT Optimization

ORB Object request broker

PML Perfectly matched layer

POSIX Portable Operating Systems for Computing Environments

PTP Provider TP

PVM Parallel virtual machine

RAM Random-access memory

RBA Radio base antenna

RM Resource management

RP Radio propagation

RSL Resource specification language

SDK Software development kit

SDSC San Diego Supercomputer Center

SIMD Simple instruction multiple data

SPMD Single program multiple data

SP-GE spectral/Gegenbauer

SP-WG spectral/waveguide

SRB Storage Resource Broker

SSL Secure sockets layer

TCP Transfer control protocol

TCP/IP Transfer control protocol/Internet protocol

TE Transverse electric

TLS Transport layer security

TM Transverse magnetic

TP Test point

TPN Test point network

URL Uniform resource locator

WAN Wide area network

WI Walfisch-Ikegami

WLAN Wireless local area network

Xinetd eXtended InterNET services daemon

Selected Bibliography

Abramowitz, M., I. Stegun, *Handbook of Mathematical Functions*, New York: Dover Publications, 1974.

Alessandri, F., M. Mongiardo, and R. Sorrentino, "New Efficient Full Wave Optimization of Microwave Circuits by the Adjoint Network Method," *IEEE Microwave Guided Wave Lett.,* Vol. 3, No. 11, November 1993, pp. 414–416.

Allen, G., E. Seidel, and J. Shalf, "Scientific Computing on the Grid," *Byte*, Spring 2002.

Baker, M., R. Buyya, and D. Laforenza, "The Grid: International Efforts in Global Computing," *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, Italy, 2000.

Berenger, J. P., "A Perfectly Matched Layer for the Absorption of EM Waves," *Journ. Comp. Phys.*, Vol. 114, 1994, pp. 185–200.

Bertoni, H., *Radio Propagation for Modern Wireless Systems*, Englewood Cliffs, NJ: Prentice Hall, 2000.

Bird, T. S., "Mutual Coupling in Finite Coplanar Rectangular Waveguides Arrays," *Electron. Lett.*, Vol. 23, Oct. 1987, pp. 1199–1201.

Bird, T. S., "Analysis of Mutual Coupling in Finite Arrays of Different Sized Rectangular Waveguides," *IEEE Trans. Antennas Propagat.*, Vol. AP-38, February 1990, pp. 166–172.

Bird, T. S., and D.G. Bateman, "Mutual Coupling Between Rotated Horns in a Ground Plane," *IEEE Trans. Antennas Propagat.*, Vol. AP-42, July 1994, pp. 1000–1006.

Booch, G., *Object-Oriented Analysis and Design (With Applications),* Redwood, CA: Benjamin-Cummings Publishing Co. Inc., 1994.

Butenhof, D. R., *Programming with POSIX Threads*, Reading, MA: Addison-Wesley, 1997.

Buyya, R., D. Abramson, and J. Giddy, "An Economic Driven Resource Management Architecture for Global Computational Power Grids," *Intl. Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000)*, Las Vegas, NV, June 2000.

Buyya, R., D.Abramson, and J. Giddy "Nimrod/G: An Architecture for a Resource Management and Scheduling in a Global Computational Grid," *4th International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*, Beijing, China, IEEE Computer Society, Los Alamitos, CA, May 2000.

Catarinucci, L., P. Palazzari, and L. Tarricone, "Human Exposure to the Near-Field of Radiobase Antennas: A Full-Wave Solution Using Parallel FDTD," *IEEE Trans. Micr. Theory Techn.*, Vol. 51, No. 3, 2003, pp. 935–941.

Chappel, D., "Examining .NET My Services," *Byte*, Spring 2002.

Chew, K.C., and V. Fusco, "A Parallel Implementation of the FDTD Algorithm," *Int. Journ. Num. Modelling*, Vol. 8, 1995, pp. 293–299.

Collin, R., *Antennas and Radiowave Propagation*, Singapore: McGraw-Hill Int. Ed., 1985.

Collin, R., *Field Theory of Guided Waves*, New York: IEEE Press, 1991.

Collmann, R. R., "Evaluation of Methods for Determining the Mobile Traffic Distribution in Cellular Radio Networks," *IEEE Transactions on Vehicular Technology,* Vol. 50, No. 6, November 2001, pp. 1629–1635.

Conciauro, G., M. Guglielmi, and R. Sorrentino, *Advanced Modal Analysis*, London: Wiley, 1999.

Decusatis, C., "Grid Computing: The Next (Really, Really) Big Thing," *Byte*, Spring 2002.

Dobrowolski, J. A., *Introduction to Computational Methods for Microwave Circuit Analysis*, Norwood, MA: Artech House, 1991.

Dongarra, J., et al., "Integrated PVM Framework Supports Heterogeneous Network Computing," *Computers in Physics*, April 1993.

Duncan, R., "A Survey of Parallel Computer Architectures," *IEEE Computer*, Vol. 23, No. 2, February 1990.

Enquist, B., and A. Majda, "ABC for the Numerical Simulation of Waves," *Math. Of Computation*, Vol. 31, 1977, pp. 629–651.

Foster, I., and C. Kesselman (Eds.), *The Grid: Blueprint for a New Computer Infrastructure,* San Francisco, CA: Morgan Kaufmann, 1999.

Foster, I., C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int. Journal of High Performance Computing Applications*, Vol. 15, No. 3, 2001.

Foster, I., et al., "Grid Services for Distributed System Integration," *IEEE Computer*, June 2002.

Gedney, S. D., "FDTD Analysis of MW Circuit Devices in High Performance Vector/Parallel Computers," *IEEE Trans. Microwave Theory Techn.,* Vol. 43, No. 10, 1995, pp. 2510–2514.

Gibson, J., *The Communication Handbook*, Boca Raton, FL: CRC Press, 1997.

Goldberg, D. E., *Genetic Algorithm in Search, Optimization and Machine-Learning*, Reading, MA: Addison Wesley, 1992.

Guiffaut, C., and K. Mahdjoubi, "A Parallel FDTD Algorithm Using the MPI Library," *IEEE Ant. Prop. Mag.,* Vol. 43, No. 2, 2001, pp. 94–103.

Hennessy, J., and D. Patterson, *Computer Organization & Design*, San Francisco: Morgan Kaufmann Publishers, 1998.

Higdon, R. L., "ABC for Difference Approximations to the Multidimensional Wave Equation," *Mathematics of Computation*, Vol. 47, 1986, pp. 437–459.

Howes, T., and M. Smith, *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol,* Macmillan Technical Publishing, 1997.

Hurley, S., "Planning Effective Cellular Mobile Radio Networks," *IEEE Transactions on Vehicular Technology,* Vol. 51, No. 2, 2002, pp. 243–253.

Jamieson, A. R., and T. E. Rozzi, "Rigorous Analysis of Cross Polarization in Flange-Mounted Rectangular Waveguide Radiators," *Electron. Lett.*, Vol. 13, November 24, 1977, pp. 742–744.

Khoshafian, S., and R. Abnous, *Object-Orientation: Concepts, Languages, Databases, User Interfaces*, New York: John Wiley & Sons, 1995.

Kitchener, D., K. Raghavan, and C. G. Parini, "Mutual Coupling in a Finite Planar Array of Rectangular Apertures," *Electronic Letters*, Vol. 23, October 21,1987, pp. 1169–1170.

Kraus, J. D., *Antennas,* New York: McGraw Hill, 1988.

Kraus, J. D., and R. J. Marefka, *Antennas for all Applications*, New York: McGraw Hill, 2002.

Kunz, K. S., and R. J. Luebbers, *The FDTD Method for Electromagnetics*, Boca Raton, FL: CRC Press Inc., 1993.

Lewin, L., *Advanced Theory of Waveguides*, London, UK: Iliffe, 1951.

Lewis, Ted G., and Hesham El-Rewini, *Introduction to Parallel Computing*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1992.

Mailloux, R. J., "Radiation and Near Field Coupling Between Two Collinear Open Ended Waveguides," *IEEE Trans. Antennas Propagat.*, Vol. AP-17, January 1969, pp. 49–55.

Mailloux, R. J., "First Order Solution for Mutual Coupling Between Waveguides Which Propagate Two Orthogonal Modes," *IEEE Trans. Antennas Propagat.*, Vol. AP-17, November 1969, pp. 740–746.

Menezes, A., P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, Boca Raton, FL: CRC Press, 1996.

Mongiardo, M., and R. Ravanelli, "Automated Design of Corrugated Feeds by the Adjoint Network Method," *IEEE Trans. Microwave Theory Tech.,* Vol. 45, May 1997, pp. 787–793.

Mongiardo, M., and T. Rozzi, "Singular Integral Equation Analysis of Flange-Mounted Rectangular Waveguide Radiators," *IEEE Trans. on Ant. and Prop*., Vol. 41, May 1993, pp. 556–565.

Mongiardo, M., L. Tarricone, and C. Tomassoni, "A Comparison of Numerical Methods for the Full-Wave Analysis of Flange Mounted Rectangular Apertures," *Int. Journal Numerical Modelling*, Vol. 13, No. 1, 2000, pp. 21–35.

Mur, G., "ABC for the FD Approximation of the Time-Domain EM Field Equations," *IEEE Trans. EM Comp*., Vol. 23, No. 4, 1981, pp. 377–382.

Nicol, J. R., C. Thomas Wilkes, and F. A. Manola, "Object Orientation in Heterogeneous Distributed Systems," *IEEE Computer*, June 1993.

Pacheco, P. S., *Parallel Programming with MPI*, San Francisco, CA: Morgan Kaufman, 1997.

Paton, N., et al., "Database Access and Integration Services on the Grid," UK e-Science Programme Technical Report Series Number UKeS-2002-03, National e-Science Centre, www.cs.man.ac.uk/grid-db/papers/dbtf.pdf.

Petri, C. A., *Kommunikation mit Automaten*, Ph.D. Thesis, University of Bonn, Germany, 1962.

Reeves, C. R. (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Press, 1992.

Saunders, S. R., *Antennas and Propagation for Wireless Communication Systems*, London: John Wiley & Sons, 1999.

Siziak, K., *Radiowave Propagation and Antennas for Personal Communications*, Norwood, MA: Artech House, 1998.

Schendel, U. "Introduction to Numerical Methods for Parallel Computers," Chichester, UK: Ellis Horwood Lim. Publishers, 1984.

Siniaris, C. G., et al., "Implementing Distributed FDTD Codes with Java Mobile Agents," *IEEE Antennas and Propagation Magazine*,Vol. 44, No. 6, December 2002, pp. 115–119.

Stalling, W., *Wireless Communications and Networks*, Englewood Cliffs, NJ: Prentice Hall, 2002.

Stutzman, W. L., and G. A. Thiele, *Antenna Theory and Design*, New York: John Wiley & Sons, 1981.

Taflove, A., *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, Norwood, MA: Artech House, 1995.

Yee, K. S., "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equation in Isotropic Media," *IEEE Trans. Antennas and Prop.*, AP-14, May 1966, pp. 302–307.

About the Authors

Luciano Tarricone is an associate professor of electromagnetic fields at the University of Lecce, Italy. He received his laurea degree (with honors) from the University of Rome, La Sapienza, Italy, and his Ph.D. from the same university, both in electronic engineering. He was a researcher at the Italian National Institute of Health in 1990 and at the IBM European Center for Scientific and Engineering Computing between 1990 and 1994. Between 1994 and 2001, he was at the University of Perugia, Italy. Since 2001, he has been with the University of Lecce.

His main research areas are supercomputing for electromagnetics, environmental electromagnetic compatibility, and CAD of microwave circuits and antennas. He authored about 160 papers in international conferences and journals and edited three volumes in the area of high-performance computing for electromagnetics.

Alessandra Esposito is a freelance consultant in the area of computer science and information technologies, with a focus on networking, Web applications, and grid applications for research in universities and small, medium, and large companies.

She received her laurea degree (with honors) in electronic engineering from the University of Naples. Between 1990 and 1994, she was with IBM Scientific Center in Rome, Italy. In 1994 and 1995 she was a system engineer for Sodalia, Trento, Italy, involved in research and development in the area of distributed systems. Since 1995, she has cooperated with several research institutions, universities, and business companies, in the framework of educational, research, and industrial projects. She authored approximately 40 papers in international conferences and journals.

Index

A

Abstract device interface (ADI), 50
Absorbing boundary conditions (ABC), 93–94, 98, 100–101, 103–104, 106–107
Adaptive mesh refinement, 7, 52
Admittance matrix, 124, 127, 128
Amdahl's law, 11
Antenna
- adaptive, 160, 161, 163
- aperture, 121, 123, 128, 129, 150, 153, 211
- array of, 121–123, 125, 128–130, 132, 133, 135–136, 138, 142–146, 151, 153, 158–161, 183
- backlobe, 159
- base station (BS), 153–173, 180
- power level, 153–155, 160, 161, 166, 168, 170–172
- radiation pattern, 129, 134–136, 142, 157, 158, 160, 162, 163
- radiobase (RBA), 95, 98
- tilting, 153, 157, 159–160, 162, 163, 166, 167, 172

APE platform, 97
Application programming interface (API), 17–18, 30, 38–39, 41, 46–49, 52, 58, 62, 80, 82–84, 88, 97–98, 109–110, 116, 174, 180
Architecture
- client-server, 12, 13
- three-tier, 13
- two-tier, 12

Arithmetic and logic unit (ALU), 2–3
Array factor, 159, 160
Array
- linear, 160
- phased, 160, 161
- planar, 158, 160

Asymmetric key encryption, 201
Asynchronous, 4, 8, 195–196
Atomic subtask (AS), 136–139, 141, 142, 144
Authentication, 20, 30, 32–33, 35–37, 47, 60, 65–66, 75, 85, 87, 200
Authorization, 20, 32–34, 37, 64–65, 87
Automation, 184–185

B

Back-end, 13–14, 27
Background, 81, 194–196
Bandwidth, 4, 11, 19, 40–41, 46–47, 89, 97, 111–114, 117, 138, 161, 183–184
Barrier, 50, 99–101, 104
Beowulf cluster, 50, 112
Berenger's ABC, 94
Bessel's function, 128
Bisection assignment policy, 140
Blocking message, 99
Blocking of a call, 157
Broadcast, 50
Brokering node, 169–173, 176, 178, 180
Browser, 12–14, 17, 27–28, 46, 52, 148

C

Cache, 40–41, 83
Cactus Code, 52
Capacity, 47, 154–155, 157, 161
Cellular system, 153–155, 158, 161
- capacity, 154, 155, 157, 161
- coverage, 154, 160, 162, 163, 166, 169
- power control, 160–162
- quality of service, 153, 157
- switching station, 154
- traffic management, 154

Certificate 34-37, 64–71, 74–75, 80, 200, 203–204
Certification authority (CA), 34–37, 60, 64, 66–71, 203–204
Characteristic admittance, 124
Cipher text, 200–203
Circuit theory, 132–134
Closed-cycle control, 160

Cluster
of computers, 4, 14–15, 18, 50, 112
of cells, 156–157
Clustering, 156
Coallocator, 38, 48, 51
Cochannel interference, 160–162
Code division multiple access (CDMA), 161–162
Collaborative engineering, 20–21
see also cooperative engineering
Collective operation, 49
Commodity Grid (CoG), 49
Commodity technologies – *see* enabling technologies
Common Gateway Interface (CGI), 12
Common Object Request Broker Architecture (CORBA), 26–27, 49, 52
Communication
diameter, 5, 6
context, 50, 103
Computational economy, 19, 51, 148
Computer aided design (CAD), 29, 130
Computer-aided engineering (CAE), 121–123, 125, 128–130, 132, 135–136, 142, 144–148, 150–153, 183
Concurrency, 8, 110, 136, 142
Concurrent composition, 142
Condor, 11, 16, 18, 38–40, 51, 53
Condor-G, 53
Confidentiality, 20, 30, 37, 60, 200–201
Connection matrix, 133, 134
Constitutive equations, 208
Consumer test point (CTP), 172–174, 180
Control processing Unit (CPU), 1–3, 8, 18–19, 21, 39–40, 44–46, 51, 53, 60, 78, 89, 92, 95, 136, 144, 146, 149
Control unit (CU), 2–3
Cooperative engineering, 117, 121–122, 136, 145, 150–151, 153, 180, 183
see also collaborative engineering
COST 231 project, 165
Cost function, 166
Coupling matrix, 129, 131, 136, 143
Courant condition, 92, 104
Cryptography, 199-200
public-key, 34, 60
Curl equation, 89–90

D

DAGman, 53
Daemon, 61, 72–73, 193–194
Data
exploration, 20–21
management, 20, 30, 48, 78, 153, 173
Database Management System (DBMS), 46, 164, 167–168, 171, 185
Data management (DM), 30, 46, 59, 63, 73, 80, 146
Decryption, 37, 200–201, 203
Delegation, 32, 35
Dielectric sphere, 115
Digital signature, 35, 200, 202–203
Directory information tree (DIT), 43
Directory service, 42–43, 59
Discovery, 19, 26, 30
Discretization
spatial, 91, 97
temporal, 92
Dispersion
equation, 92, 208
error, 92
numerical, 90, 92–93
Distinguished name (DN), 32–34, 43–44, 64–65, 68–70
Distributed
memory architecture, 4–5, 7, 11
programming, 7
Distributed Component Object Model (DCOM), 26
Distributed Computing Environment (DCE), 26
Distributed Parallel Storage System (DPSS), 20, 46
Domain
decomposition, 108, 139, 144
partitioning, 99–100, 111, 139-140
Domain name system, (DNS), 59, 67
Downlink connection, 160–161
Dynamically Updated Request Online Coallocator (DUROC), 48

E

Efficiency, 10–11, 97–98, 111–112, 143
Encapsulation, 25–26, 29–30
Encapsulation module (ENC), 165, 168, 170–172, 178
Encryption, 36–37, 200–201
Enquist and Majda theory, 94
Environment variable, 10, 61–62, 87, 111, 141, 196–198
Exec, 193
Expanding function, 125, 127, 131
Extended Access Control List (EACL), 70

EXtended InterNET services daemon (xinetd), 72–74, 194

F

Far-field approximation, 135, 212
Fault tolerance, 16
Feeder, 129
File system, 19, 44, 46, 58, 81, 87, 188–189, 191, 194–195
File transfer protocol (ftp), 19, 30, 46, 59, 62, 78, 80, 145, 173
First tier interference, 156
Finite difference time domain (FDTD),
  differential approach, 91
  excitations in, 93
  integral approach, 91
  mesh , 92, 209
  parallel, 96, 98, 113, 116–117, 144, 183
Flynn's taxonomy, 3
Fork, 39, 58, 149, 193
Fourier transformation, 125–127, 135, 211
Frii's formula, 165
Frequency reassignment, 157
Frequency reuse, 154–157
Frontend tier, 13
Fully qualified domain name (FQDN), 57, 60–61, 77, 84, 87, 109, 175–176, 178

G

Gatekeeper , 38–39, 58–59, 64–67, 71–72, 149–150
Gegenbauer polynomial, 125, 127–128
Genetic algorithm, 144, 166
Geographical Information System (GIS), 164, 167–168, 171–172
Global System for Mobile Communications (GSM), 95, 98–99, 154–155
Globus Access to Secondary Storage (GASS), 40–41, 59, 80–85, 174–175, 177, 180
  functions:
  globus_gass_fclose, 83–84, 174, 178
  globus_gass_fopen, 83–84, 174, 178
  globus_module_activate, 83–84, 174, 178
  globus_module_deactivate, 83–84, 174, 178
Globus Packaging Toolkit (GPT), 60–62
Globus Resource Allocation Manager (GRAM), 38–39, 41, 48–49, 58–59, 64, 72
Globus Security Infrastructure (GSI), 30–33, 35–39, 47–48, 60, 63
Globus Toolkit (GT)
  commands
  globus-gass-server, 81, 82, 84–85, 175
  globus-job-run, 75–77
  globusrun, 77–79, 147–148
  globus-url-copy, 80–82, 147–148
  grid-info-search, 79
  grid-proxy-destroy, 75
  grid-proxy-init, 75, 81, 85
  flavor, 63, 85, 110
Google, 52
Green's function, 123, 124, 126, 127, 210, 211
Grid
  architecture, 17,19
  services, 184
Grid architecture for computational economy (GRACE), 19, 51
Grid database access and management (GDAM), 184–185
GridFTP, 42–43, 59
Grid index information system (GIIS), 42–43, 59
Grid middleware (GM), 17–20, 23–24, 27, 30, 52
Grid Resource Information Service (GRIS), 42–43, 45, 48, 59, 78–79

H

Half-wave dipole, 115, 116, 158, 160
Handover, 158
Handshaking, 158
Helmholtz equation, 94, 208–210
High-performance computing (HPC), 1, 4, 14–15, 20–21, 51, 89, 117, 121–122, 144–145, 151, 166, 170, 180, 183–184
High throughput Ccomputing (HTC), 20–21, 53
Home directory, 67, 71, 194, 197–198
Horn, 129–130, 132–134, 136, 137, 142, 143
Host-node model, 9
Hub, 154–155
Human-antenna interaction, 89, 121, 153, 183
Hypertext mark-up language (HTML), 11–14
Hypertext transfer protocol (HTTP), 12–13, 19, 62, 81
Hypertext transfer protocol over SSL (HTTPS), 81, 84

I

Information management (IM), 18, 72, 78
  see also information services

Information services (IS), 30, 42, 59, 63, 146, 148
see also information management
Information model, 42–43, 45
Inheritance, 25
Integration
path, 126
point, 131, 132, 139
Integrity, 7, 30, 37, 47, 60, 200
Interface definition language (IDL), 27

J

Java
Language, 12-13, 22, 26–28, 49, 52, 116
Virtual machine, 27–28, 116
Servlet, 27
JavaBeans, 13, 26
Jini, 26–28
Java mobile agents (JMA), 13, 26–29, 116
Job
manager, 38–41, 58
scheduling, 11

K

Kerberos, 18, 20, 47

L

Latency, 10, 47
Layer
fabric, 17–18, 23
middleware, 17–18, 23
application, 17–18, 23
Leap-frog scheme, 91
Legion, 30
Level, *see* layer
Lewin-transformation, 125–128
Lightweight directory access protocol (LDAP), 43–44, 46, 59, 80
Linux
account, 34, 64–67, 74, 194, 198
Linux commands
adduser, 194
cd, 62, 71, 86, 177, 189, 191
chmod, 68, 71, 192
cp, 68, 69, 71, 86, 177, 190
gunzip, 62, 86, 109, 190–191
gzip, 177, 190, 197
ln, 69, 190
make, 86, 177
mkdir, 86, 176, 189
mv, 69, 190
rm, 190, 195
rmdir, 189
tar, 62, 86, 109, 177, 191, 197
Linux
kernel, 187–188, 192, 194
shell, 61, 64, 74, 170, 173, 178, 179, 188, 194-198
Load balancing, 11, 19, 87, 99, 109–110, 117, 123, 139, 141, 144, 146
Loading Sharing Facility (LSF), 11, 18, 38–39, 51
Loadleveler, 21
Local area network (LAN), 5, 14–16, 60, 113, 115
Ludwig third definition, 135, 142

M

Main block (MB), 136–138, 141, 143–144
Man-in-the-middle attack, 34, 203–204
Massively parallel processor (MPP), 4–5, 14, 50, 112–113
Master-worker, 9
Maxwell equations, 24, 89–91, 93, 95, 207–208
Meta-application, 20–21, 183–184
Metacomputer, 23, 30
Metacomputing directory service (MDS), 42–46, 59, 72–73
Message passing, 4, 7–8, 10, 49–50, 85, 88, 98, 111–113, 116, 141
Metallic flange, 122–123, 134, 142,
Method of moments (MOM), 121, 136
Middle tier, 13–14
Mobile Agent, 26, 116
Message-passing interface (MPI), 8–91, 49–50, 85, 89, 98–103, 108–116, 141
Message-passing interface (MPI) functions:
MPI_Barrier, 99–100
MPI_Comm_rank , 99, 103
MPI_Comm_size, 103–104
MPI_Init, 103
MPI_Recv, 105, 108
MPI_Send, 105, 108
Mode
dominant, 123
higher order, 123
TE, 123, 124, 129, 135, 209, 210
TM, 123, 124, 129, 135, 209, 210
Mode-matching (MM), 129, 136
MPICH, 9, 38, 50, 85–87, 109–115, 123, 141–142, 144–145
MPICH-G2, 9, 50, 85, 89, 109–115, 117

Multilevel parallelism, 138, 139, 142, 144
Multiprocessor, 7–10, 15, 89, 110, 111
Multiple instruction multiple data (MIMD), 3–4, 96, 98, 108, 117, 123, 142, 144
Multithreading, 112
Multithreaded application, 8
Multitier, 13
Mur's ABC, 94, 98, 100–101, 103, 106–107
Mutual authentication, 32–33, 35–37, 60, 65–66
Mutual coupling, 122, 123, 129, 130, 135, 136, 138, 139, 143, 147

N

Near field, 95, 98, 159, 213
Network Time Protocol (NTP), 60, 87, 176–177
Nimrod-G, 51, 148–150

O

Object Management group (OMG), 26–27
Object-oriented (OO), 4, 23–30, 53, 141
  abstraction, 25, 30
  class, 25
  component, 13–14, 25–27
  container, 26
Object request broker (ORB), 27
Object Web, 52
Okumura-Hata model (OH), 165
Open-cycle control, 160
OpenLDAP, 59
OpenMP, 9–10
OpenSSL, 59–60, 66, 69–71
Optimization, 47, 121, 122, 134, 136, 144, 153, 157, 158, 163, 164, 166, 170

P

Paging, 158
Parallel
  architecture, 3–4, 10, 145
  programming, 7
Parallel virtual machine (PVM), 8–9, 49
Path loss, 154
Peer-to-peer, 14
Perfectly matched layer (PML), 94
Petri net, 123, 137–139, 141–142
Phase velocity, 92, 209
Pipe, 195–197
Plane-wave representation, 211
Plain text, 195, 200–203
Plan file, 149–150
PML ABC, 94
  See also Berenger's ABC
Point-to-point communication, 49, 99
Polarization
  cross, 123
  x, 127
  y, 127
Polymorphism, 25, 30
Port
  number, 58, 64, 72, 73, 81, 83, 85, 175
  physical, 132
  serial, 169, 171
Portable Batch System (PBS), 18
Portable Operating Systems for Computing Environments (POSIX), 110, 187
Private key, 34–37, 67–71, 75, 200-203
Process
  parent, 8
  group, 49
  topology, 50
Profile script, 61, 74, 198
Profiling, 136–139, 141
Propagation constant, 124
Provider test point (PTP), 172–180
Proxy, 36–37, 60, 75
Pthreads, 110, 112
Public-key, 33–37, 60, 67, 201–204

R

Radiopropagation model (RP), 153, 163–166, 169, 171, 172, 178
Rectangular
  aperture, 123, 124, 128, 129, 150
  waveguide, 122, 123, 135, 143, 209
Recursive bisection, 139
Registration, 19
Relation, 38, 40, 41
Replica, 20, 46, 47
Resource management (RM), 30, 38, 51, 58–59, 63, 72, 75, 85, 109–110, 146
Resource specification language (RSL), 38–42, 59, 77–79, 85, 109, 146–148
Roaming, 161–163
Root account, 61, 66–68, 176, 194

S

San Diego Supercomputer Center (SDSC), 56
Scattering matrix, 128–134, 136–137
Sectorization, 155–157, 159–160
Secure copy (SCP), 80
Secure sockets layer (SSL), 33–36, 60, 66, 81

Security, 16, 20, 31-37, 81, 87, 121-123, 135, 151, 169–170, 199–200
Sensor network, 163, 165
SETI@home, 14
Sequential composition, 142
Shared memory
  architecture, 4, 7, 50
  programming, 8–10
Signal-to-noise ratio, 158, 160, 161
Simple instruction multiple data (SIMD), 3, 96–98, 108, 117, 142
Single program multiple data (SPMD), 9, 98, 141–142, 144
Single sign-on, 32, 35–37, 75
Socket, 145
Software development kit (SDK), 58, 62
Software engineering, 24, 29, 121, 139
Sources in FDTD
  current, 93
  hard, 93, 106
  soft, 93
Speed-up, 10-11, 95–96, 105, 111–112, 116, 141, 143–145, 183
Standard error, 193, 195–196
Standard input, 193, 195–196
Standard output, 193, 195–196
Storage Resource Broker (SRB), 20, 46, 48
Symmetric key algorithm, 200–201
System call, 39, 58, 149, 188–189, 192–193

T

Tabu search, 166
TAO, 97
Transfer Control Protocol/Internet Protocol (TCP/IP), 57, 60–61, 87
Thread, 8, 40, 58, 63, 110–111, 117
Thread-safe, 111
Three-tier, 13
Topology, 5, 101–102, 107
  Cartesian, 101–102, 106–107
  logical, 6–7
  mesh, 5–7
  ring , 5
  toroidal, 97
  tree, 6–7
  virtual process, 50
Test point (TP), 164–165, 168, 171–172
Test point network (TPN), 164–165, 168, 171, 175
Two tier, 12

U

Universal Mobile Telecommunication System (UMTS), 154
Uniform resource locator (URL), 41, 78, 81, 83
Uplink connection, 160–161

V

Vector potential
electric **F**, 209
magnetic **A**, 208
Visual KAP, 10
Von Neumann
approach, 92
model , 2–3

W

Walfisch-Ikegami model (WA), 165,
Wave number, 124
Web
  computing, 11–14
  architecture, 11–14
  projects, 51–52
  portal, 52, 184–185
Webflow, 52
Wide area network (WAN), 16, 50, 60
Wi-Fi, 161
Wireless local area network (WLAN), 161
Wireless network planning, 162, 163, 170, 182, 183

Y

Yee's
algorithm, 89–90
cell, 90–92

Recent Titles in the Artech House Electromagnetic Analysis Series

Tapan K. Sarkar, Series Editor

Advances in Computational Electrodynamics: The Finite-Difference Time-Domain Method, Allen Taflove, editor

Analysis Methods for Electromagnetic Wave Problems, Volume 2, Eikichi Yamashita, editor

Analytical Modeling in Applied Electromagnetics, Sergei Tretyakov

Applications of Neural Networks in Electromagnetics, Christos Christodoulou and Michael Georgiopoulos

CFDTD: Conformal Finite-Difference Time-Domain Maxwell's Equations Solver, Software and User's Guide, Wenhua Yu and Raj Mittra

The CG-FFT Method: Application of Signal Processing Techniques to Electromagnetics, Manuel F. Cátedra, et al.

Computational Electrodynamics: The Finite-Difference Time-Domain Method, Second Edition, Allen Taflove and Susan C. Hagness

Electromagnetic Waves in Chiral and Bi-Isotropic Media, I. V. Lindell, et al.

Engineering Applications of the Modulated Scatterer Technique, Jean-Charles Bolomey and Fred E. Gardiol

Fast and Efficient Algorithms in Computational Electromagnetics, Weng Cho Chew, et al., editors

Fresnel Zones in Wireless Links, Zone Plate Lenses and Antennas, Hristo D. Hristov

Grid Computing for Electromagnetics, Luciano Tarricone and Alessandra Esposito

Iterative and Self-Adaptive Finite-Elements in Electromagnetic Modeling, Magdalena Salazar-Palma, et al.

Quick Finite Elements for Electromagnetic Waves, Giuseppe Pelosi, Roberto Coccioli, and Stefano Selleri

Understanding Electromagnetic Scattering Using the Moment Method: A Practical Approach, Randy Bancroft

Wavelet Applications in Engineering Electromagnetics, Tapan K. Sarkar, Magdalena Salazar-Palma, and Michael C. Wicks

For further information on these and other Artech House titles,
including previously considered out-of-print books now available through our
In-Print-Forever® (IPF®) program, contact:

Artech House
685 Canton Street
Norwood, MA 02062
Phone: 781-769-9750
Fax: 781-769-6334
e-mail: artech@artechhouse.com

Artech House
46 Gillingham Street
London SW1V 1AH UK
Phone: +44 (0)20 7596-8750
Fax: +44 (0)20 7630 0166
e-mail: artech-uk@artechhouse.com

Find us on the World Wide Web at:
www.artechhouse.com