

# PARALLEL PROGRAMMING IN C WITH MPI AND OPENMP

## 并行程序设计 C、MPI与OpenMP



Michael J. Quinn 著



清华大学出版社

清华大学出版社 中国计算机教育出版集团 中国计算机教育出版集团

# PARALLEL PROGRAMMING IN C WITH MPI AND OPENMP

## 并行程序设计 C、MPI与OpenMP



Michael J. Quinn 著



清华大学出版社

大学计算机教育国外著名教材系列(影印版)

Parallel Programming  
in C with MPI and OpenMP

# 并行程序设计 C、MPI 与 OpenMP

Michael J. Quinn

清华大学出版社  
北 京

Michael J. Quinn

**Parallel Programming in C with MPI and OpenMP**

EISBN: 0-07-282256-2

Copyright © 2004 by The McGraw-Hill Companies, Inc.

Original language published by The McGraw-Hill Companies, Inc. All Rights reserved. No part of this publication may be reproduced or distributed by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

Authorized English language edition jointly published by McGraw-Hill Education (Asia) Co. and Tsinghua University Press. This edition is authorized for sale only to the educational and training institutions, and within the territory of the People's Republic of China (excluding Hong Kong, Macao SAR and Taiwan). Unauthorized export of this edition is a violation of the Copyright Act. Violation of this Law is subject to Civil and Criminal Penalties.

本书英文影印版由清华大学出版社和美国麦格劳-希尔教育出版(亚洲)公司合作出版。此版本仅限在中华人民共和国境内(不包括中国香港、澳门特别行政区及中国台湾地区)针对教育及培训机构之销售。未经许可之出口,视为违反著作权法,将受法律之制裁。  
未经出版者预先书面许可,不得以任何方式复制或抄袭本书的任何部分。

北京市版权局著作权合同登记号 图字: 01-2005-0654 号

版权所有, 翻印必究。举报电话: 010-62782989 13501256678 13801310933

本书封面贴有 McGraw-Hill 公司防伪标签, 无标签者不得销售。

**图书在版编目(CIP)数据**

并行程序设计: C、MPI 与 OpenMP 英文/(美)奎因(Quinn, M. J.)著. —影印本. —北京: 清华大学出版社, 2005. 8

(大学计算机教育国外著名教材系列)

ISBN 7-302-11157-X

I. 并… II. 奎… III. 并行程序—程序设计—高等学校—教材—英文 IV. TP311.11

中国版本图书馆 CIP 数据核字(2005)第 058944 号

出 版 者: 清华大学出版社 地 址: 北京清华大学学研大厦

<http://www.tup.com.cn> 邮 编: 100084

社总机: 010-62770175 客户服务: 010-62776969

印 刷 者: 北京牛山世兴印刷厂

装 订 者: 三河市金元装订厂

发 行 者: 新华书店总店北京发行所

开 本: 148 × 210 印张: 16.875

版 次: 2005 年 8 月第 1 版 2005 年 8 月第 1 次印刷

书 号: ISBN 7-302-11157-X/TP · 7373

印 数: 1 ~ 3000

定 价: 33.00 元

# 出版说明

进入 21 世纪, 世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的竞争。谁拥有大量高素质的人才, 谁就能在竞争中取得优势。高等教育, 作为培养高素质人才的事业, 必然受到高度重视。目前我国高等教育的教材更新较慢, 为了加快教材的更新频率, 教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从 1996 年开始, 与国外著名出版公司合作, 影印出版了“大学计算机教育丛书(影印版)”等一系列引进图书, 受到国内读者的欢迎和支持。跨入 21 世纪, 我们本着为我国高等教育教材建设服务的初衷, 在已有的基础上, 进一步扩大选题内容, 改变图书开本尺寸, 一如既往地请有关专家挑选适用于我国高等本科及研究生计算机教育的国外经典教材或著名教材, 组成本套“大学计算机教育国外著名教材系列(影印版)”, 以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材, 以利我们把“大学计算机教育国外著名教材系列(影印版)”做得更好, 更适合高校师生的需要。

清华大学出版社

# PREFACE

---

**T**his book is a practical introduction to parallel programming in C using the MPI (Message Passing Interface) library and the OpenMP application programming interface. It is targeted to upper-division undergraduate students, beginning graduate students, and computer professionals learning this material on their own. It assumes the reader has a good background in C programming and has had an introductory class in the analysis of algorithms.

Fortran programmers interested in parallel programming can also benefit from this text. While the examples in the book are in C, the underlying concepts of parallel programming with MPI and OpenMP are essentially the same for both C and Fortran programmers.

In the past twenty years I have taught parallel programming to hundreds of undergraduate and graduate students. In the process I have learned a great deal about the sorts of problems people encounter when they begin “thinking in parallel” and writing parallel programs. Students benefit from seeing programs designed and implemented step by step. My philosophy is to introduce new functionality “just in time.” As much as possible, every new concept appears in the context of solving a design, implementation, or analysis problem. When you see the symbol



in a page margin, you’ll know I’m presenting a key concept.

The first two chapters explain when and why parallel computing began and gives a high-level overview of parallel architectures. Chapter 3 presents Foster’s parallel algorithm design methodology and shows how it is used through several case studies. Chapters 4, 5, 6, 8, and 9 demonstrate how to use the design methodology to develop MPI programs that solve a series of progressively more difficult programming problems. The 27 MPI functions presented in these chapters are a robust enough subset to implement parallel programs for a wide variety of applications. These chapters also introduce functions that simplify matrix and vector I/O. The source code for this I/O library appears in Appendix B.

The programs of Chapters 4, 5, 6, and 8 have been benchmarked on a commodity cluster of microprocessors, and these results appear in the text. Because new generations of microprocessors appear much faster than books can be produced, readers will observe that the processors are several generations old. The point of presenting the results is not to amaze the reader with the speed of the computations. Rather, the purpose of the benchmarking is to demonstrate that knowledge of the latency and bandwidth of the interconnection network, combined with information about the performance of a sequential program, are often sufficient to allow reasonably accurate predictions of the performance of a parallel program.

Chapter 7 focuses on four metrics for analyzing and predicting the performance of parallel systems: Amdahl's Law, Gustafson-Barsis' Law, the Karp-Flatt metric, and the isoefficiency metric.

Chapters 10–16 provide additional examples of how to analyze a problem and design a good parallel algorithm to solve it. At this point the development of MPI programs implementing the parallel algorithms is left to the reader. I present Monte Carlo methods and the challenges associated with parallel random number generation. Later chapters present a variety of key algorithms: matrix multiplication, Gaussian elimination, the conjugate gradient method, finite difference methods, sorting, the fast Fourier transform, backtrack search, branch-and-bound search, and alpha-beta search.

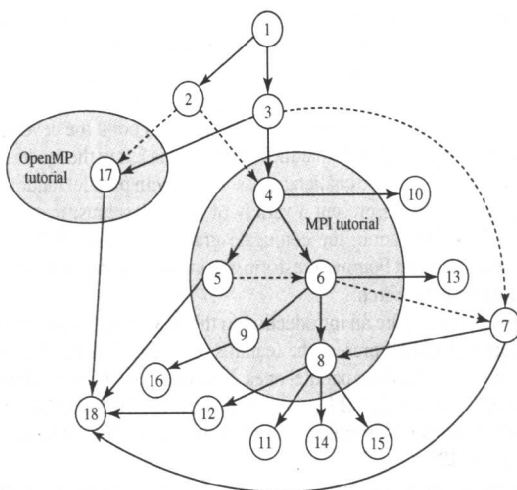
Chapters 17 and 18 are an introduction to the new shared-memory programming standard OpenMP. I present the features of OpenMP as needed to convert sequential code segments into parallel ones. I use two case studies to demonstrate the process of transforming MPI programs into hybrid MPI/OpenMP programs that can exhibit higher performance on multiprocessor clusters than programs based solely on MPI.

This book has more than enough material for a one-semester course in parallel programming. While parallel programming is more demanding than typical programming, it is also more rewarding. Even with a teacher's instruction and support, most students are unnerved at the prospect of harnessing multiple processors to perform a single task. However, this fear is transformed into a feeling of genuine accomplishment when they see their debugged programs run much faster than "ordinary" C programs. For this reason, programming assignments should play a central role in the course.

Fortunately, parallel computers are more accessible than ever. If a commercial parallel computer is not available, it is a straightforward task to build a small cluster out of a few PCs, networking equipment, and free software.

Figure P.1 illustrates the precedence relations among the chapters. A solid arrow from A to B indicates chapter B depends heavily upon material presented in chapter A. A dashed arrow from A to B indicates a weak dependence. If you cover the chapters in numerical order, you will satisfy all of these precedences. However, if you would like your students to start programming in C with MPI as quickly as possible, you may wish to skip Chapter 2 or only cover one or two sections of it. If you wish to focus on numerical algorithms, you may wish to skip Chapter 5 and introduce students to the function `MPI_Bcast` in another way. If you would like to start by having your students programming Monte Carlo algorithms, you can jump to Chapter 10 immediately after Chapter 4. If you want to cover OpenMP before MPI, you can jump to Chapter 17 after Chapter 3.

I thank everyone at McGraw-Hill who helped me create this book, especially Betsy Jones, Michelle Flomenhoft, and Kay Brimeyer. Thank you for your sponsorship, encouragement, and assistance. I also appreciate the help provided by Maggie Murphy and the rest of the composers at Interactive Composition Corporation.



**Figure P.1** Dependences among the chapters. A solid arrow indicates a strong dependence; a dashed arrow indicates a weak dependence.

I am indebted to the reviewers who carefully read the manuscript, correcting errors, pointing out weak spots, and suggesting additional topics. My thanks to: A. P. W. Bohm, Colorado State University; Thomas Cormen, Dartmouth College; Narsingh Deo, University of Central Florida; Philip J. Hatcher, University of New Hampshire; Nickolas S. Jovanovic, University of Arkansas at Little Rock; Dinesh Mehta, Colorado School of Mines; Zina Ben Miled, Indiana University-Purdue University, Indianapolis; Paul E. Plassman, Pennsylvania State University; Quinn O. Snell, Brigham Young University; Ashok Srinivasan, Florida State University; Xian-He Sun, Illinois Institute of Technology; Virgil Wallentine, Kansas State University; Bob Weems, University of Texas at Arlington; Kay Zemoudeh, California State University-San Bernardino; and Jun Zhang, University of Kentucky.

Many people at Oregon State University also lent me a hand. Rubin Landau and Henri Jansen helped me understand Monte Carlo algorithms and the detailed balance condition, respectively. Students Charles Sauerbier and Bernd Michael Kelm suggested questions that made their way into the text. Tim Budd showed me how to incorporate PostScript figures into LaTeX documents. Jalal Haddad provided technical support. Thank you for your help!

Finally, I am grateful to my wife, Victoria, for encouraging me to get back into textbook writing. Thanks for the inspiring Christmas present: *Chicken Soup for the Writer's Soul: Stories to Open the Heart and Rekindle the Spirit of Writers*.

**Michael J. Quinn**  
Corvallis, Oregon



# CONTENTS

---

Preface xiv

## CHAPTER 1

### Motivation and History 1

- 1.1 Introduction 1
- 1.2 Modern Scientific Method 3
- 1.3 Evolution of Supercomputing 4
- 1.4 Modern Parallel Computers 5
  - 1.4.1 *The Cosmic Cube* 6
  - 1.4.2 *Commercial Parallel Computers* 6
  - 1.4.3 *Beowulf* 7
  - 1.4.4 *Advanced Strategic Computing Initiative* 8
- 1.5 Seeking Concurrency 9
  - 1.5.1 *Data Dependence Graphs* 9
  - 1.5.2 *Data Parallelism* 10
  - 1.5.3 *Functional Parallelism* 10
  - 1.5.4 *Pipelining* 12
  - 1.5.5 *Size Considerations* 13
- 1.6 Data Clustering 14
- 1.7 Programming Parallel Computers 17
  - 1.7.1 *Extend a Compiler* 17
  - 1.7.2 *Extend a Sequential Programming Language* 18
  - 1.7.3 *Add a Parallel Programming Layer* 19
  - 1.7.4 *Create a Parallel Language* 19
  - 1.7.5 *Current Status* 21
- 1.8 Summary 21
- 1.9 Key Terms 22
- 1.10 Bibliographic Notes 22
- 1.11 Exercises 23

## CHAPTER 2

### Parallel Architectures 27

- 2.1 Introduction 27
- 2.2 Interconnection Networks 28
  - 2.2.1 *Shared versus Switched Media* 28
  - 2.2.2 *Switch Network Topologies* 29
  - 2.2.3 *2-D Mesh Network* 29
  - 2.2.4 *Binary Tree Network* 30
  - 2.2.5 *Hypertree Network* 31
  - 2.2.6 *Butterfly Network* 32
  - 2.2.7 *Hypercube Network* 33
  - 2.2.8 *Shuffle-exchange Network* 35
  - 2.2.9 *Summary* 36
- 2.3 Processor Arrays 37
  - 2.3.1 *Architecture and Data-parallel Operations* 37
  - 2.3.2 *Processor Array Performance* 39
  - 2.3.3 *Processor Interconnection Network* 40
  - 2.3.4 *Enabling and Disabling Processors* 40
  - 2.3.5 *Additional Architectural Features* 42
  - 2.3.6 *Shortcomings of Processor Arrays* 42
- 2.4 Multiprocessors 43
  - 2.4.1 *Centralized Multiprocessors* 43
  - 2.4.2 *Distributed Multiprocessors* 45
- 2.5 Multicomputers 49
  - 2.5.1 *Asymmetrical Multicomputers* 49
  - 2.5.2 *Symmetrical Multicomputers* 51
  - 2.5.3 *Which Model Is Best for a Commodity Cluster?* 52
  - 2.5.4 *Differences between Clusters and Networks of Workstations* 53
- 2.6 Flynn's Taxonomy 54
  - 2.6.1 *SISD* 54
  - 2.6.2 *SIMD* 55

- 2.6.3 *MISD* 55
- 2.6.4 *MIMD* 56
- 2.7 Summary 58
- 2.8 Key Terms 59
- 2.9 Bibliographic Notes 59
- 2.10 Exercises 60

## CHAPTER 3

### Parallel Algorithm Design 63

- 3.1 Introduction 63
- 3.2 The Task/Channel Model 63
- 3.3 Foster's Design Methodology 64
  - 3.3.1 *Partitioning* 65
  - 3.3.2 *Communication* 67
  - 3.3.3 *Agglomeration* 68
  - 3.3.4 *Mapping* 70
- 3.4 Boundary Value Problem 73
  - 3.4.1 *Introduction* 73
  - 3.4.2 *Partitioning* 75
  - 3.4.3 *Communication* 75
  - 3.4.4 *Agglomeration and Mapping* 76
  - 3.4.5 *Analysis* 76
- 3.5 Finding the Maximum 77
  - 3.5.1 *Introduction* 77
  - 3.5.2 *Partitioning* 77
  - 3.5.3 *Communication* 77
  - 3.5.4 *Agglomeration and Mapping* 81
  - 3.5.5 *Analysis* 82
- 3.6 The  $n$ -Body Problem 82
  - 3.6.1 *Introduction* 82
  - 3.6.2 *Partitioning* 83
  - 3.6.3 *Communication* 83
  - 3.6.4 *Agglomeration and Mapping* 85
  - 3.6.5 *Analysis* 85
- 3.7 Adding Data Input 86
  - 3.7.1 *Introduction* 86
  - 3.7.2 *Communication* 87
  - 3.7.3 *Analysis* 88
- 3.8 Summary 89

- 3.9 Key Terms 90
- 3.10 Bibliographic Notes 90
- 3.11 Exercises 90

## CHAPTER 4

### Message-Passing Programming 93

- 4.1 Introduction 93
- 4.2 The Message-Passing Model 94
- 4.3 The Message-Passing Interface 95
- 4.4 Circuit Satisfiability 96
  - 4.4.1 *Function MPI\_Init* 99
  - 4.4.2 *Functions MPI\_Comm\_rank and MPI\_Comm\_size* 99
  - 4.4.3 *Function MPI\_Finalize* 101
  - 4.4.4 *Compiling MPI Programs* 102
  - 4.4.5 *Running MPI Programs* 102
- 4.5 Introducing Collective Communication 104
  - 4.5.1 *Function MPI\_Reduce* 105
- 4.6 Benchmarking Parallel Performance 108
  - 4.6.1 *Functions MPI\_Wtime and MPI\_Wtick* 108
  - 4.6.2 *Function MPI\_Barrier* 108
- 4.7 Summary 110
- 4.8 Key Terms 110
- 4.9 Bibliographic Notes 110
- 4.10 Exercises 111

## CHAPTER 5

### The Sieve of Eratosthenes 115

- 5.1 Introduction 115
- 5.2 Sequential Algorithm 115
- 5.3 Sources of Parallelism 117
- 5.4 Data Decomposition Options 117
  - 5.4.1 *Interleaved Data Decomposition* 118
  - 5.4.2 *Block Data Decomposition* 118
  - 5.4.3 *Block Decomposition Macros* 120
  - 5.4.4 *Local Index versus Global Index* 120

#### 5.4.5 *Ramifications of Block Decomposition* 121

- 5.5 Developing the Parallel Algorithm 121
  - 5.5.1 *Function MPI\_Bcast* 122
- 5.6 Analysis of Parallel Sieve Algorithm 122
- 5.7 Documenting the Parallel Program 123
- 5.8 Benchmarking 128
- 5.9 Improvements 129
  - 5.9.1 *Delete Even Integers* 129
  - 5.9.2 *Eliminate Broadcast* 130
  - 5.9.3 *Reorganize Loops* 131
  - 5.9.4 *Benchmarking* 131
- 5.10 Summary 133
- 5.11 Key Terms 134
- 5.12 Bibliographic Notes 134
- 5.13 Exercises 134

### CHAPTER 6

#### **Floyd's Algorithm** 137

- 6.1 Introduction 137
- 6.2 The All-Pairs Shortest-Path Problem 137
- 6.3 Creating Arrays at Run Time 139
- 6.4 Designing the Parallel Algorithm 140
  - 6.4.1 *Partitioning* 140
  - 6.4.2 *Communication* 141
  - 6.4.3 *Agglomeration and Mapping* 142
  - 6.4.4 *Matrix Input/Output* 143
- 6.5 Point-to-Point Communication 145
  - 6.5.1 *Function MPI\_Send* 146
  - 6.5.2 *Function MPI\_Recv* 147
  - 6.5.3 *Deadlock* 148
- 6.6 Documenting the Parallel Program 149
- 6.7 Analysis and Benchmarking 151
- 6.8 Summary 154
- 6.9 Key Terms 154
- 6.10 Bibliographic Notes 154
- 6.11 Exercises 154

### CHAPTER 7

#### **Performance Analysis** 159

- 7.1 Introduction 159
- 7.2 Speedup and Efficiency 159
- 7.3 Amdahl's Law 161
  - 7.3.1 *Limitations of Amdahl's Law* 164
  - 7.3.2 *The Amdahl Effect* 164
- 7.4 Gustafson-Barsis's Law 164
- 7.5 The Karp-Flatt Metric 167
- 7.6 The Isoefficiency Metric 170
- 7.7 Summary 174
- 7.8 Key Terms 175
- 7.9 Bibliographic Notes 175
- 7.10 Exercises 176

### CHAPTER 8

#### **Matrix-Vector Multiplication** 178

- 8.1 Introduction 178
- 8.2 Sequential Algorithm 179
- 8.3 Data Decomposition Options 180
- 8.4 Rowwise Block-Striped Decomposition 181
  - 8.4.1 *Design and Analysis* 181
  - 8.4.2 *Replicating a Block-Mapped Vector* 183
  - 8.4.3 *Function MPI\_Allgatherv* 184
  - 8.4.4 *Replicated Vector Input/Output* 186
  - 8.4.5 *Documenting the Parallel Program* 187
  - 8.4.6 *Benchmarking* 187
- 8.5 Columnwise Block-Striped Decomposition 189
  - 8.5.1 *Design and Analysis* 189
  - 8.5.2 *Reading a Columnwise Block-Striped Matrix* 191
  - 8.5.3 *Function MPI\_Scatterv* 191
  - 8.5.4 *Printing a Columnwise Block-Striped Matrix* 193
  - 8.5.5 *Function MPI\_Gatherv* 193
  - 8.5.6 *Distributing Partial Results* 195

8.5.7	<i>Function MPI_Alltoallv</i>	195
8.5.8	<i>Documenting the Parallel Program</i>	196
8.5.9	<i>Benchmarking</i>	198
8.6	<i>Checkerboard Block Decomposition</i>	199
8.6.1	<i>Design and Analysis</i>	199
8.6.2	<i>Creating a Communicator</i>	202
8.6.3	<i>Function MPI_Dims_create</i>	203
8.6.4	<i>Function MPI_Cart_create</i>	204
8.6.5	<i>Reading a Checkerboard Matrix</i>	205
8.6.6	<i>Function MPI_Cart_rank</i>	205
8.6.7	<i>Function MPI_Cart_coords</i>	207
8.6.8	<i>Function MPI_Comm_split</i>	207
8.6.9	<i>Benchmarking</i>	208
8.7	<i>Summary</i>	210
8.8	<i>Key Terms</i>	211
8.9	<i>Bibliographic Notes</i>	211
8.10	<i>Exercises</i>	211

## CHAPTER 9

### Document Classification 216

9.1	<i>Introduction</i>	216
9.2	<i>Parallel Algorithm Design</i>	217
9.2.1	<i>Partitioning and Communication</i>	217
9.2.2	<i>Agglomeration and Mapping</i>	217
9.2.3	<i>Manager/Worker Paradigm</i>	218
9.2.4	<i>Manager Process</i>	219
9.2.5	<i>Function MPI_Abort</i>	220
9.2.6	<i>Worker Process</i>	221
9.2.7	<i>Creating a Workers-only Communicator</i>	223
9.3	<i>Nonblocking Communications</i>	223
9.3.1	<i>Manager's Communication</i>	224
9.3.2	<i>Function MPI_Irecv</i>	224
9.3.3	<i>Function MPI_Wait</i>	225
9.3.4	<i>Workers' Communications</i>	225
9.3.5	<i>Function MPI_Isend</i>	225
9.3.6	<i>Function MPI_Probe</i>	225
9.3.7	<i>Function MPI_Get_count</i>	226
9.4	<i>Documenting the Parallel Program</i>	226
9.5	<i>Enhancements</i>	232

9.5.1	<i>Assigning Groups of Documents</i>	232
9.5.2	<i>Pipelining</i>	232
9.5.3	<i>Function MPI_Testsome</i>	234
9.6	<i>Summary</i>	235
9.7	<i>Key Terms</i>	236
9.8	<i>Bibliographic Notes</i>	236
9.9	<i>Exercises</i>	236

## CHAPTER 10

### Monte Carlo Methods 239

10.1	<i>Introduction</i>	239
10.1.1	<i>Why Monte Carlo Works</i>	240
10.1.2	<i>Monte Carlo and Parallel Computing</i>	243
10.2	<i>Sequential Random Number Generators</i>	243
10.2.1	<i>Linear Congruential</i>	244
10.2.2	<i>Lagged Fibonacci</i>	245
10.3	<i>Parallel Random Number Generators</i>	245
10.3.1	<i>Manager-Worker Method</i>	246
10.3.2	<i>Leapfrog Method</i>	246
10.3.3	<i>Sequence Splitting</i>	247
10.3.4	<i>Parameterization</i>	248
10.4	<i>Other Random Number Distributions</i>	248
10.4.1	<i>Inverse Cumulative Distribution Function Transformation</i>	249
10.4.2	<i>Box-Muller Transformation</i>	250
10.4.3	<i>The Rejection Method</i>	251
10.5	<i>Case Studies</i>	253
10.5.1	<i>Neutron Transport</i>	253
10.5.2	<i>Temperature at a Point Inside a 2-D Plate</i>	255
10.5.3	<i>Two-Dimensional Ising Model</i>	257
10.5.4	<i>Room Assignment Problem</i>	259
10.5.5	<i>Parking Garage</i>	262
10.5.6	<i>Traffic Circle</i>	264
10.6	<i>Summary</i>	268
10.7	<i>Key Terms</i>	269
10.8	<i>Bibliographic Notes</i>	269
10.9	<i>Exercises</i>	270

**CHAPTER 11****Matrix Multiplication 273**

- 11.1 Introduction 273
- 11.2 Sequential Matrix Multiplication 274
  - 11.2.1 *Iterative, Row-Oriented Algorithm* 274
  - 11.2.2 *Recursive, Block-Oriented Algorithm* 275
- 11.3 Rowwise Block-Striped Parallel Algorithm 277
  - 11.3.1 *Identifying Primitive Tasks* 277
  - 11.3.2 *Agglomeration* 278
  - 11.3.3 *Communication and Further Agglomeration* 279
  - 11.3.4 *Analysis* 279
- 11.4 Cannon's Algorithm 281
  - 11.4.1 *Agglomeration* 281
  - 11.4.2 *Communication* 283
  - 11.4.3 *Analysis* 284
- 11.5 Summary 286
- 11.6 Key Terms 287
- 11.7 Bibliographic Notes 287
- 11.8 Exercises 287

**CHAPTER 12****Solving Linear Systems 290**

- 12.1 Introduction 290
- 12.2 Terminology 291
- 12.3 Back Substitution 292
  - 12.3.1 *Sequential Algorithm* 292
  - 12.3.2 *Row-Oriented Parallel Algorithm* 293
  - 12.3.3 *Column-Oriented Parallel Algorithm* 295
  - 12.3.4 *Comparison* 295
- 12.4 Gaussian Elimination 296
  - 12.4.1 *Sequential Algorithm* 296
  - 12.4.2 *Parallel Algorithms* 298
  - 12.4.3 *Row-Oriented Algorithm* 299
  - 12.4.4 *Column-Oriented Algorithm* 303

- 12.4.5 *Comparison* 303
- 12.4.6 *Pipelined, Row-Oriented Algorithm* 304

- 12.5 Iterative Methods 306
- 12.6 The Conjugate Gradient Method 309
  - 12.6.1 *Sequential Algorithm* 309
  - 12.6.2 *Parallel Implementation* 310
- 12.7 Summary 313
- 12.8 Key Terms 314
- 12.9 Bibliographic Notes 314
- 12.10 Exercises 314

**CHAPTER 13****Finite Difference Methods 318**

- 13.1 Introduction 318
- 13.2 Partial Differential Equations 320
  - 13.2.1 *Categorizing PDEs* 320
  - 13.2.2 *Difference Quotients* 321
- 13.3 Vibrating String 322
  - 13.3.1 *Deriving Equations* 322
  - 13.3.2 *Deriving the Sequential Program* 323
  - 13.3.3 *Parallel Program Design* 324
  - 13.3.4 *Isoefficiency Analysis* 327
  - 13.3.5 *Replicating Computations* 327
- 13.4 Steady-State Heat Distribution 329
  - 13.4.1 *Deriving Equations* 329
  - 13.4.2 *Deriving the Sequential Program* 330
  - 13.4.3 *Parallel Program Design* 332
  - 13.4.4 *Isoefficiency Analysis* 332
  - 13.4.5 *Implementation Details* 334
- 13.5 Summary 334
- 13.6 Key Terms 335
- 13.7 Bibliographic Notes 335
- 13.8 Exercises 335

**CHAPTER 14****Sorting 338**

- 14.1 Introduction 338
- 14.2 Quicksort 339

- 14.3 A Parallel Quicksort Algorithm 340
  - 14.3.1 Definition of Sorted 340
  - 14.3.2 Algorithm Development 341
  - 14.3.3 Analysis 341
- 14.4 Hyperquicksort 343
  - 14.4.1 Algorithm Description 343
  - 14.4.2 Isoefficiency Analysis 345
- 14.5 Parallel Sorting by Regular Sampling 346
  - 14.5.1 Algorithm Description 346
  - 14.5.2 Isoefficiency Analysis 347
- 14.6 Summary 349
- 14.7 Key Terms 349
- 14.8 Bibliographic Notes 350
- 14.9 Exercises 350

## CHAPTER 15

### The Fast Fourier Transform 353

- 15.1 Introduction 353
- 15.2 Fourier Analysis 353
- 15.3 The Discrete Fourier Transform 355
  - 15.3.1 Inverse Discrete Fourier Transform 357
  - 15.3.2 Sample Application: Polynomial Multiplication 357
- 15.4 The Fast Fourier Transform 360
- 15.5 Parallel Program Design 363
  - 15.5.1 Partitioning and Communication 363
  - 15.5.2 Agglomeration and Mapping 365
  - 15.5.3 Isoefficiency Analysis 365
- 15.6 Summary 367
- 15.7 Key Terms 367
- 15.8 Bibliographic Notes 367
- 15.9 Exercises 367

## CHAPTER 16

### Combinatorial Search 369

- 16.1 Introduction 369
- 16.2 Divide and Conquer 370

- 16.3 Backtrack Search 371
  - 16.3.1 Example 371
  - 16.3.2 Time and Space Complexity 374
- 16.4 Parallel Backtrack Search 374
- 16.5 Distributed Termination Detection 377
- 16.6 Branch and Bound 380
  - 16.6.1 Example 380
  - 16.6.2 Sequential Algorithm 382
  - 16.6.3 Analysis 385
- 16.7 Parallel Branch and Bound 385
  - 16.7.1 Storing and Sharing Unexamined Subproblems 386
  - 16.7.2 Efficiency 387
  - 16.7.3 Halting Conditions 387
- 16.8 Searching Game Trees 388
  - 16.8.1 Minimax Algorithm 388
  - 16.8.2 Alpha-Beta Pruning 392
  - 16.8.3 Enhancements to Alpha-Beta Pruning 395
- 16.9 Parallel Alpha-Beta Search 395
  - 16.9.1 Parallel Aspiration Search 396
  - 16.9.2 Parallel Subtree Evaluation 396
  - 16.9.3 Distributed Tree Search 397
- 16.10 Summary 399
- 16.11 Key Terms 400
- 16.12 Bibliographic Notes 400
- 16.13 Exercises 401

## CHAPTER 17

### Shared-Memory Programming 404

- 17.1 Introduction 404
- 17.2 The Shared-Memory Model 405
- 17.3 Parallel for Loops 407
  - 17.3.1 parallel for Pragma 408
  - 17.3.2 Function omp\_get\_num\_procs 410
  - 17.3.3 Function omp\_set\_num\_threads 410
- 17.4 Declaring Private Variables 410
  - 17.4.1 privateClause 411

- 17.4.2 *firstprivate Clause* 412
- 17.4.3 *lastprivate Clause* 412
- 17.5 Critical Sections 413
  - 17.5.1 *critical Pragma* 415
- 17.6 Reductions 415
- 17.7 Performance Improvements 417
  - 17.7.1 *Inverting Loops* 417
  - 17.7.2 *Conditionally Executing Loops* 418
  - 17.7.3 *Scheduling Loops* 419
- 17.8 More General Data Parallelism 421
  - 17.8.1 *parallel Pragma* 422
  - 17.8.2 *Function omp\_get\_thread\_num* 423
  - 17.8.3 *Function omp\_get\_num\_threads* 425
  - 17.8.4 *for Pragma* 425
  - 17.8.5 *single Pragma* 427
  - 17.8.6 *nowait Clause* 427
- 17.9 Functional Parallelism 428
  - 17.9.1 *parallel sections Pragma* 429
  - 17.9.2 *section Pragma* 429
  - 17.9.3 *sections Pragma* 429
- 17.10 Summary 430
- 17.11 Key Terms 432
- 17.12 Bibliographic Notes 432
- 17.13 Exercises 433

## CHAPTER 18

### Combining MPI and OpenMP 436

- 18.1 Introduction 436
- 18.2 Conjugate Gradient Method 438
  - 18.2.1 *MPI Program* 438
  - 18.2.2 *Functional Profiling* 442
  - 18.2.3 *Parallelizing Function matrix\_vector\_product* 442
  - 18.2.4 *Benchmarking* 443
- 18.3 Jacobi Method 444
  - 18.3.1 *Profiling MPI Program* 444

- 18.3.2 *Parallelizing Function find\_steady\_state* 444
- 18.3.3 *Benchmarking* 446

## 18.4 Summary 448

## 18.5 Exercises 448

## APPENDIX A

### MPI Functions 450

## APPENDIX B

### Utility Functions 485

#### B.1 Header File *MyMPI.h* 485

#### B.2 Source File *MyMPI.c* 486

## APPENDIX C

### Debugging MPI Programs 505

#### C.1 Introduction 505

#### C.2 Typical Bugs in MPI Programs 505

##### C.2.1 *Bugs Resulting in Deadlock* 505

##### C.2.2 *Bugs Resulting in Incorrect Results* 506

##### C.2.3 *Advantages of Collective Communications* 507

#### C.3 Practical Debugging Strategies 507

## APPENDIX D

### Review of Complex Numbers 509

## APPENDIX E

### OpenMP Functions 513

## Bibliography 515

## 1

# Motivation and History

*Well done is quickly done.*  
Caesar Augustus

## 1.1 INTRODUCTION

Are you one of those people for whom “fast” isn’t fast enough? Today’s workstations are about a hundred times faster than those made just a decade ago, but some computational scientists and engineers need even more speed. They make great simplifications to the problems they are solving and still must wait hours, days, or even weeks for their programs to finish running.

Faster computers let you tackle larger computations. Suppose you can afford to wait overnight for your program to produce a result. If your program suddenly ran 10 times faster, previously out-of-reach computations would now be within your grasp. You could produce in 15 hours an answer that previously required nearly a week to generate.

Of course, you *could* simply wait for CPUs to get faster. In about five years single CPUs will be 10 times faster than they are today (a consequence of Moore’s Law). On the other hand, if you can afford to wait five years, you must not be in that much of a hurry! Parallel computing is a proven way to get higher performance *now*.

### *What’s parallel computing?*

**Parallel computing** is the use of a parallel computer to reduce the time needed to solve a single computational problem. Parallel computing is now considered a standard way for computational scientists and engineers to solve problems in areas as diverse as galactic evolution, climate modeling, aircraft design, and molecular dynamics.





*What's a parallel computer?*

A **parallel computer** is a multiple-processor computer system supporting parallel programming. Two important categories of parallel computers are multi-computers and centralized multiprocessors.

As its name implies, a **multicomputer** is a parallel computer constructed out of multiple computers and an interconnection network. The processors on different computers interact by passing messages to each other.

In contrast, a **centralized multiprocessor** (also called a **symmetrical multiprocessor** or **SMP**) is a more highly integrated system in which all CPUs share access to a single global memory. This shared memory supports communication and synchronization among processors.

We'll study centralized multiprocessors, multicomputers, and other parallel computer architectures in Chapter 2.

*What's parallel programming?*

**Parallel programming** is programming in a language that allows you to explicitly indicate how different portions of the computation may be executed concurrently by different processors. We'll discuss various kinds of parallel programming languages in more detail near the end of this chapter.

*Is parallel programming really necessary?*

A lot of research has been invested in the development of compiler technology that would allow ordinary Fortran 77 or C programs to be translated into codes that would execute with good efficiency on parallel computers with large numbers of processors. This is a very difficult problem, and while many experimental parallelizing<sup>1</sup> compilers have been developed, at the present time commercial systems are still in their infancy. The alternative is for you to write your own parallel programs.

*Why should I program using MPI and OpenMP?*

MPI (Message Passing Interface) is a standard specification for message-passing libraries. Libraries meeting the standard are available on virtually every parallel computer system. Free libraries are also available in case you want to run MPI on a network of workstations or a parallel computer built out of commodity components (PCs and switches). If you develop programs using MPI, you will be able to reuse them when you get access to a newer, faster parallel computer.

Increasingly, parallel computers are being constructed out of symmetrical multiprocessors. Within each SMP, the CPUs have a shared address space. While MPI is a perfectly satisfactory way for processors in different SMPs to communicate with each other, OpenMP is a better way for processors within a single SMP

<sup>1</sup>parallelize verb: to make parallel.