PARALLEL PROGRAMMING
TECHNIQUES AND APPLICATIONS USING
NETWORKED WORKSTATIONS AND
PARALLEL COMPUTERS

2nd Edition

BARRY WILKINSON
University of North Carolina at Charlotte
Western Carolina University

MICHAEL ALLEN
University of North Carolina at Charlotte

PEARSON Prentice Hall

Upper Saddle River, NJ 07458
To my wife, Wendy,
and my daughter, Johanna
Barry Wilkinson

To my wife, Bonnie
Michael Allen
Preface

The purpose of this text is to introduce parallel programming techniques. Parallel programming is programming multiple computers, or computers with multiple internal processors, to solve a problem at a greater computational speed than is possible with a single computer. It also offers the opportunity to tackle larger problems, that is, problems with more computational steps or larger memory requirements, the latter because multiple computers and multiprocessor systems often have more total memory than a single computer. In this text, we concentrate upon the use of multiple computers that communicate with one another by sending messages; hence the term message-passing parallel programming. The computers we use can be different types (PC, SUN, SGI, etc.) but must be interconnected, and a software environment must be present for message passing between computers. Suitable computers (either already in a network or capable of being interconnected) are very widely available as the basic computing platform for students, so that it is usually not necessary to acquire a specially designed multiprocessor system. Several software tools are available for message-passing parallel programming, notably several implementations of MPI, which are all freely available. Such software can also be used on specially designed multiprocessor systems should these systems be available for use. So far as practicable, we discuss techniques and applications in a system-independent fashion.

Second Edition. Since the publication of the first edition of this book, the use of interconnected computers as a high-performance computing platform has become widespread. The term “cluster computing” has come to be used to describe this type of computing. Often the computers used in a cluster are “commodity” computers, that is, low-cost personal computers as used in the home and office. Although the focus of this text, using multiple computers and processors for high-performance computing, has not been changed, we have revised our introductory chapter, Chapter 1, to take into account the move towards
commodity clusters and away from specially designed, self-contained, multiprocessors. In
the first edition, we described both PVM and MPI and provided an appendix for each.
However, only one would normally be used in the classroom. In the second edition, we have
deleted specific details of PVM from the text because MPI is now a widely adopted
standard and provides for much more powerful mechanisms. PVM can still be used if one
wishes, and we still provide support for it on our home page.

Message-passing programming has some disadvantages, notably the need for the
programmer to specify explicitly where and when the message passing should occur in the
program and what to send. Data has to be sent to those computers that require the data
through relatively slow messages. Some have compared this type of programming to
assembly language programming, that is, programming using the internal language of the
computer, a very low-level and tedious way of programming which is not done except
under very specific circumstances. An alternative programming model is the shared
memory model. In the first edition, shared memory programming was covered for
computers with multiple internal processors and a common shared memory. Such shared
memory multiprocessors have now become cost-effective and common, especially dual-
and quad-processor systems. Thread programming was described using Pthreads. Shared
memory programming remains in the second edition and with significant new material
added including performance aspects of shared memory programming and a section on
OpenMP, a thread-based standard for shared memory programming at a higher level than
Pthreads. Any broad-ranging course on practical parallel programming would include
shared memory programming, and having some experience with OpenMP is very desir-
able. A new appendix is added on OpenMP. OpenMP compilers are available at low cost
to educational institutions.

With the focus of using clusters, a major new chapter has been added on shared
memory programming on clusters. The shared memory model can be employed on a cluster
with appropriate distributed shared memory (DSM) software. Distributed shared memory
programming attempts to obtain the advantages of the scalability of clusters and the elegance
of shared memory. Software is freely available to provide the DSM environment, and we
shall also show that students can write their own DSM systems (we have had several done
so). We should point out that there are performance issues with DSM. The performance of
software DSM cannot be expected to be as good as true shared memory programming on a
shared memory multiprocessor. But a large, scalable shared memory multiprocessor is much
more expensive than a commodity cluster.

Other changes made for the second edition are related to programming on clusters.
New material is added in Chapter 6 on partially synchronous computations, which are par-
ticularly important in clusters where synchronization is expensive in time and should be
avoided. We have revised and added to Chapter 10 on sorting to include other sorting algo-
rithms for clusters. We have added to the analysis of the algorithms in the first part of the
book to include the computation/communication ratio because this is important to message-
passing computing. Extra problems have been added. The appendix on parallel computa-
tional models has been removed to maintain a reasonable page count.

The first edition of the text was described as course text primarily for an undergradu-
ate-level parallel programming course. However, we found that some institutions also
used the text as a graduate-level course textbook. We have also used the material for both
ten undergraduate-level and graduate-level courses, and it is suitable for beginning
graduate-level courses. For a graduate-level course, more advanced materials, for example, DSM implementation and fast Fourier transforms, would be covered and more demanding programming projects chosen.

**Structure of Materials.** As with the first edition, the text is divided into two parts. Part I now consists of Chapters 1 to 9, and Part II now consists of Chapters 10 to 13. In Part I, the basic techniques of parallel programming are developed. In Chapter 1, the concept of parallel computers is now described with more emphasis on clusters. Chapter 2 describes message-passing routines in general and particular software (MPI). Evaluating the performance of message-passing programs, both theoretically and in practice, is discussed. Chapter 3 describes the ideal problem for making parallel the embarrassingly parallel computation where the problem can be divided into independent parts. In fact, important applications can be parallelized in this fashion. Chapters 4, 5, 6, and 7 describe various programming strategies (partitioning and divide and conquer, pipelining, synchronous computations, asynchronous computations, and load balancing). These chapters of Part I cover all the essential aspects of parallel programming with the emphasis on message-passing and using simple problems to demonstrate techniques. The techniques themselves, however, can be applied to a wide range of problems. Sample code is usually given first as sequential code and then as parallel pseudocode. Often, the underlying algorithm is already parallel in nature and the sequential version has “unnaturally” serialized it using loops. Of course, some algorithms have to be reformulated for efficient parallel solution, and this reformulation may not be immediately apparent. Chapter 8 describes shared memory programming and includes Pthreads, an IEEE standard system that is widely available, and OpenMP. There is also a significant new section on timing and performance issues. The new chapter on distributed shared memory programming has been placed after the shared memory chapter to complete Part I, and the subsequent chapters have been renumbered.

Many parallel computing problems have specially developed algorithms, and in Part II problem-specific algorithms are studied in both non-numeric and numeric domains. For Part II, some mathematical concepts are needed, such as matrices. Topics covered in Part II include sorting (Chapter 10), numerical algorithms, matrix multiplication, linear equations, partial differential equations (Chapter 11), image processing (Chapter 12), and searching and optimization (Chapter 13). Image processing is particularly suitable for parallelization and is included as an interesting application with significant potential for projects. The fast Fourier transform is discussed in the context of image processing. This important transform is also used in many other areas, including signal processing and voice recognition.

A large selection of “real-life” problems drawn from practical situations is presented at the end of each chapter. These problems require no specialized mathematical knowledge and are a unique aspect of this text. They develop skills in the use of parallel programming techniques rather than simply teaching how to solve specific problems, such as sorting numbers or multiplying matrices.

**Prerequisites.** The prerequisite for studying Part I is a knowledge of sequential programming, as may be learned from using the C language. The parallel pseudocode in the text uses C-like assignment statements and control flow statements. However, students with only a knowledge of Java will have no difficulty in understanding the pseudocode,
because syntax of the statements is similar to that of Java. Part I can be studied immediately after basic sequential programming has been mastered. Many assignments here can be attempted without specialized mathematical knowledge. If MPI is used for the assignments, programs are usually written in C or C++ calling MPI message-passing library routines. The descriptions of the specific library calls needed are given in Appendix A. It is possible to use Java, although students with only a knowledge of Java should not have any difficulty in writing their assignments in C/C++.

In Part II, the sorting chapter assumes that the student has covered sequential sorting in a data structure or sequential programming course. The numerical algorithms chapter requires the mathematical background that would be expected of senior computer science or engineering undergraduates.

Course Structure. The instructor has some flexibility in the presentation of the materials. Not everything need be covered. In fact, it is usually not possible to cover the whole book in a single semester. A selection of topics from Part I would be suitable as an addition to a normal sequential programming class. We have introduced our first-year students to parallel programming in this way. In that context, the text is a supplement to a sequential programming course text. All of Part I and selected parts of Part II together are suitable as a more advanced undergraduate or beginning graduate-level parallel programming/computing course, and we use the text in that manner.

Home Page. A Web site has been developed for this book as an aid to students and instructors. It can be found at www.cs.uncc.edu/par_prog. Included at this site are extensive Web pages to help students learn how to compile and run parallel programs. Sample programs are provided for a simple initial assignment to check the software environment. The Web site has been completely redesigned during the preparation of the second edition to include step-by-step instructions for students using navigation buttons. Details of DSM programming are also provided. The new Instructor’s Manual is available to instructors, and gives MPI solutions. The original solutions manual gave PVM solutions and is still available. The solutions manuals are available electronically from the authors. A very extensive set of slides is available from the home page.

Acknowledgments. The first edition of this text was the direct outcome of a National Science Foundation grant awarded to the authors at the University of North Carolina at Charlotte to introduce parallel programming in the first college year. Without the support of the late Dr. M. Mulder, program director at the National Science Foundation, we would not have been able to pursue the ideas presented in the text. A number of graduate students worked on the original project. Mr. Uday Kamath produced the original solutions manual.

We should like to record our thanks to James Robinson, the departmental system administrator who established our local workstation cluster, without which we would not have been able to conduct the work. We should also like to thank the many students at UNC Charlotte who took our classes and helped us refine the material over many years. This
included “teleclasses” in which the materials for the first edition were classroom tested in a unique setting. The teleclasses were broadcast to several North Carolina universities, including UNC Asheville, UNC Greensboro, UNC Wilmington, and North Carolina State University, in addition to UNC Charlotte. Professor Mladen Vouk of North Carolina State University, apart from presenting an expert guest lecture for us, set up an impressive Web page that included “real audio” of our lectures and “automatically turning” slides. (These lectures can be viewed from a link from our home page.) Professor John Board of Duke University and Professor Jan Prins of UNC Chapel Hill also kindly made guest-expert presentations to classes. A parallel programming course based upon the material in this text was also given at the Universidad Nacional de San Luis in Argentina by kind invitation of Professor Raul Gallard.

The National Science Foundation has continued to support our work on cluster computing, and this helped us develop the second edition. A National Science Foundation grant was awarded to us to develop distributed shared memory tools and educational materials. Chapter 9, on distributed shared memory programming, describes the work. Subsequently, the National Science Foundation awarded us a grant to conduct a three-day workshop at UNC Charlotte in July 2001 on teaching cluster computing, which enabled us to further refine our materials for this book. We wish to record our appreciation to Dr. Andrew Bernat, program director at the National Science Foundation, for his continuing support. He suggested the cluster computing workshop at Charlotte. This workshop was attended by 18 faculty from around the United States. It led to another three-day workshop on teaching cluster computing at Gujarat University, Ahmedabad, India, in December 2001, this time by invitation of the IEEE Task Force on Cluster Computing (TFCC), in association with the IEEE Computer Society, India. The workshop was attended by about 40 faculty. We are also deeply in the debt to several people involved in the workshop, and especially to Mr. Rajkumar Buyya, chairman of the IEEE Computer Society Task Force on Cluster Computing who suggested it. We are also very grateful to Prentice Hall for providing copies of our textbook to free of charge to everyone who attended the workshops.

We have continued to test the materials with student audiences at UNC Charlotte and elsewhere (including the University of Massachusetts, Boston, while on leave of absence). A number of UNC-Charlotte students worked with us on projects during the development of the second edition. The new Web page for this edition was developed by Omar Lahbabi and further refined by Sari Ansari, both undergraduate students. The solutions manual in MPI was done by Thad Drum and Gabriel Medin, also undergraduate students at UNC-Charlotte.

We would like to express our continuing appreciation to Petra Recter, senior acquisitions editor at Prentice Hall, who supported us throughout the development of the second edition. Reviewers provided us with very helpful advice, especially one anonymous reviewer whose strong views made us revisit many aspects of this book, thereby definitely improving the material.

Finally, we wish to thank the many people who contacted us about the first edition, providing us with corrections and suggestions. We maintained an on-line errata list which was useful as the book went through reprints. All the corrections from the first edition have

---

2National Science Foundation grant “Parallel Programming on Workstation Clusters,” ref. DUE 995030.
3National Science Foundation grant supplement for a cluster computing workshop, ref. DUE 0119508.
been incorporated into the second edition. An on-line errata list will be maintained again for the second edition with a link from the home page. We always appreciate being contacted with comments or corrections. Please send comments and corrections to us at wilkinson@email.wcu.edu (Barry Wilkinson) or cma@uncc.edu (Michael Allen).

BARRY WILKINSON
Western Carolina University

MICHAEL ALLEN
University of North Carolina, Charlotte
Barry Wilkinson is a full professor in the Department of Computer Science at the University of North Carolina at Charlotte, and also holds a faculty position at Western Carolina University. He previously held faculty positions at Brighton Polytechnic, England (1984–87), the State University of New York, College at New Paltz (1983–84), University College, Cardiff, Wales (1976–83), and the University of Aston, England (1973–76). From 1969 to 1970, he worked on process control computer systems at Ferranti Ltd. He is the author of Computer Peripherals (with D. Horrocks, Hodder and Stoughton, 1980, 2nd ed. 1987), Digital System Design (Prentice Hall, 1987, 2nd ed. 1992), Computer Architecture Design and Performance (Prentice Hall 1991, 2nd ed. 1996), and The Essence of Digital Design (Prentice Hall, 1997). In addition to these books, he has published many papers in major computer journals. He received a B.S. degree in electrical engineering (with first-class honors) from the University of Salford in 1969, and M.S. and Ph.D. degrees from the University of Manchester (Department of Computer Science), England, in 1971 and 1974, respectively. He has been a senior member of the IEEE since 1983 and received an IEEE Computer Society Certificate of Appreciation in 2001 for his work on the IEEE Task Force on Cluster Computing (TFCC) education program.

Michael Allen is a full professor in the Department of Computer Science at the University of North Carolina at Charlotte. He previously held faculty positions as an associate and full professor in the Electrical Engineering Department at the University of North Carolina at Charlotte (1974–85), and as an instructor and an assistant professor in the Electrical Engineering Department at the State University of New York at Buffalo (1968–74). From 1985 to 1987, he was on leave from the University of North Carolina at Charlotte while serving as the president and chairman of DataSpan, Inc. Additional industry experience includes electronics design and software systems development for Eastman Kodak, Sylvania Electronics, Bell of Pennsylvania, Wachovia Bank, and numerous other firms. He received B.S. and M.S. degrees in Electrical Engineering from Carnegie Mellon University in 1964 and 1965, respectively, and a Ph.D. from the State University of New York at Buffalo in 1968.
Contents

Preface v

About the Authors xi

PART I BASIC TECHNIQUES 1

CHAPTER 1 PARALLEL COMPUTERS 3

1.1 The Demand for Computational Speed 3

1.2 Potential for Increased Computational Speed 6
   Speedup Factor 6
   What Is the Maximum Speedup? 8
   Message-Passing Computations 13

1.3 Types of Parallel Computers 13
   Shared Memory Multiprocessor System 14
   Message-Passing Multicomputer 16
   Distributed Shared Memory 24
   MIMD and SIMD Classifications 25

1.4 Cluster Computing 26
   Interconnected Computers as a Computing Platform 26
   Cluster Configurations 32
   Setting Up a Dedicated “Beowulf Style” Cluster 36

1.5 Summary 38

Further Reading 38
CHAPTER 2 MESSAGE-PASSING COMPUTING

2.1 Basics of Message-Passing Programming 42
   Programming Options 42
   Process Creation 43
   Message-Passing Routines 46

2.2 Using a Cluster of Computers 51
   Software Tools 51
   MPI 52
   Pseudocode Constructs 60

2.3 Evaluating Parallel Programs 62
   Equations for Parallel Execution Time 62
   Time Complexity 65
   Comments on Asymptotic Analysis 68
   Communication Time of Broadcast/Gather 69

2.4 Debugging and Evaluating Parallel Programs Empirically 70
   Low-Level Debugging 70
   Visualization Tools 71
   Debugging Strategies 72
   Evaluating Programs 72
   Comments on Optimizing Parallel Code 74

2.5 Summary 75

Further Reading 75

Bibliography 76

Problems 77

CHAPTER 3 EMBARRASSINGLY PARALLEL COMPUTATIONS 79

3.1 Ideal Parallel Computation 79

3.2 Embarrassingly Parallel Examples 81
   Geometrical Transformations of Images 81
   Mandelbrot Set 86
   Monte Carlo Methods 93

3.3 Summary 98

Further Reading 99

Bibliography 99

Problems 100
CHAPTER 4    PARTITIONING AND DIVIDE-AND-CONQUER STRATEGIES

4.1    Partitioning  106
Partitioning Strategies 106
Divide and Conquer 111
M-ary Divide and Conquer 116

4.2    Partitioning and Divide-and-Conquer Examples  117
Sorting Using Bucket Sort 117
Numerical Integration 122
N-Body Problem 126

4.3    Summary  131
Further Reading 131
Bibliography 132
Problems 133

CHAPTER 5    PIPELINED COMPUTATIONS

5.1    Pipeline Technique  140

5.2    Computing Platform for Pipelined Applications  144

5.3    Pipeline Program Examples  145
Adding Numbers 145
Sorting Numbers 148
Prime Number Generation 152
Solving a System of Linear Equations — Special Case 154

5.4    Summary  157
Further Reading 158
Bibliography 158
Problems 158

CHAPTER 6    SYNCHRONOUS COMPUTATIONS

6.1    Synchronization  163
Barrier 163
Counter Implementation 165
Tree Implementation 167
Butterfly Barrier 167
Local Synchronization 169
Deadlock 169
6.2 Synchronized Computations 170
   Data Parallel Computations 170
   Synchronous Iteration 173

6.3 Synchronous Iteration Program Examples 174
   Solving a System of Linear Equations by Iteration 174
   Heat-Distribution Problem 180
   Cellular Automata 190

6.4 Partially Synchronous Methods 191

6.5 Summary 193

Further Reading 193

Bibliography 193

Problems 194

CHAPTER 7 LOAD BALANCING AND TERMINATION DETECTION 201

7.1 Load Balancing 201

7.2 Dynamic Load Balancing 203
   Centralized Dynamic Load Balancing 204
   Decentralized Dynamic Load Balancing 205
   Load Balancing Using a Line Structure 207

7.3 Distributed Termination Detection Algorithms 210
   Termination Conditions 210
   Using Acknowledgment Messages 211
   Ring Termination Algorithms 212
   Fixed Energy Distributed Termination Algorithm 214

7.4 Program Example 214
   Shortest-Path Problem 214
   Graph Representation 215
   Searching a Graph 217

7.5 Summary 223

Further Reading 223

Bibliography 224

Problems 225

CHAPTER 8 PROGRAMMING WITH SHARED MEMORY 230

8.1 Shared Memory Multiprocessors 230

8.2 Constructs for Specifying Parallelism 232
   Creating Concurrent Processes 232
   Threads 234
8.3 Sharing Data  239
   Creating Shared Data 239
   Accessing Shared Data 239

8.4 Parallel Programming Languages and Constructs  247
   Languages 247
   Language Constructs 248
   Dependency Analysis 250

8.5 OpenMP  253

8.6 Performance Issues  258
   Shared Data Access 258
   Shared Memory Synchronization 260
   Sequential Consistency 262

8.7 Program Examples  265
   UNIX Processes 265
   Pthreads Example 268
   Java Example 270

8.8 Summary  271
   Further Reading 272
   Bibliography 272
   Problems 273

CHAPTER 9  DISTRIBUTED SHARED MEMORY SYSTEMS
          AND PROGRAMMING  279

9.1 Distributed Shared Memory  279
9.2 Implementing Distributed Shared Memory  281
   Software DSM Systems 281
   Hardware DSM Implementation 282
   Managing Shared Data 283
   Multiple Reader/Single Writer Policy in a Page-Based System 284

9.3 Achieving Consistent Memory in a DSM System  284

9.4 Distributed Shared Memory Programming Primitives  286
   Process Creation 286
   Shared Data Creation 287
   Shared Data Access 287
   Synchronization Accesses 288
   Features to Improve Performance 288

9.5 Distributed Shared Memory Programming  290

9.6 Implementing a Simple DSM system  291
   User Interface Using Classes and Methods 291
   Basic Shared-Variable Implementation 292
   Overlapping Data Groups 295
PART II  ALGORITHMS AND APPLICATIONS

CHAPTER 10  SORTING ALGORITHMS

10.1 General  303
   Sorting  303
   Potential Speedup  304

10.2 Compare-and-Exchange Sorting Algorithms  304
   Compare and Exchange  304
   Bubble Sort and Odd-Even Transposition Sort  307
   Mergesort  311
   Quicksort  313
   Odd-Even Mergesort  316
   Bitonic Mergesort  317

10.3 Sorting on Specific Networks  320
   Two-Dimensional Sorting  321
   Quicksort on a Hypercube  323

10.4 Other Sorting Algorithms  327
   Rank Sort  327
   Counting Sort  330
   Radix Sort  331
   Sample Sort  333
   Implementing Sorting Algorithms on Clusters  333

10.5 Summary  335

Further Reading 335
Bibliography 336
Problems 337

CHAPTER 11  NUMERICAL ALGORITHMS

11.1 Matrices — A Review  340
   Matrix Addition  340
   Matrix Multiplication  341
   Matrix-Vector Multiplication  341
   Relationship of Matrices to Linear Equations  342
11.2 Implementing Matrix Multiplication 342
Algorithm 342
Direct Implementation 343
Recursive Implementation 346
Mesh Implementation 348
Other Matrix Multiplication Methods 352
11.3 Solving a System of Linear Equations 352
Linear Equations 352
Gaussian Elimination 353
Parallel Implementation 354
11.4 Iterative Methods 356
Jacobi Iteration 357
Faster Convergence Methods 360
11.5 Summary 365
Further Reading 365
Bibliography 365
Problems 366

CHAPTER 12 IMAGE PROCESSING 370
12.1 Low-level Image Processing 370
12.2 Point Processing 372
12.3 Histogram 373
12.4 Smoothing, Sharpening, and Noise Reduction 374
Mean 374
Median 375
Weighted Masks 377
12.5 Edge Detection 379
Gradient and Magnitude 379
Edge-Detection Masks 380
12.6 The Hough Transform 383
12.7 Transformation into the Frequency Domain 387
Fourier Series 387
Fourier Transform 388
Fourier Transforms in Image Processing 389
Parallelizing the Discrete Fourier Transform Algorithm 391
Fast Fourier Transform 395
12.8 Summary 400
Further Reading 401
CHAPTER 13 SEARCHING AND OPTIMIZATION 406

13.1 Applications and Techniques 406

13.2 Branch-and-Bound Search 407
   Sequential Branch and Bound 407
   Parallel Branch and Bound 409

13.3 Genetic Algorithms 411
   Evolution and Genetic Algorithms 411
   Sequential Genetic Algorithms 413
   Initial Population 413
   Selection Process 415
   Offspring Production 416
   Variations 418
   Termination Conditions 418
   Parallel Genetic Algorithms 419

13.4 Successive Refinement 423

13.5 Hill Climbing 424
   Banking Application 425
   Hill Climbing in a Banking Application 427
   Parallelization 428

13.6 Summary 428
   Further Reading 428
   Bibliography 429
   Problems 430

APPENDIX A BASIC MPI ROUTINES 437

APPENDIX B BASIC PTHREAD ROUTINES 444

APPENDIX C OPENMP DIRECTIVES, LIBRARY FUNCTIONS, AND ENVIRONMENT VARIABLES 449

INDEX 460