



INTRODUCTION TO

PARALLEL COMPUTING

DESIGN AND ANALYSIS OF ALGORITHMS

VIPIN KUMAR

ANANTH GRAMA

ANSHUL GUPTA

GEORGE KARYPIS

Introduction to Parallel Computing

Design and Analysis of Algorithms

Vipin Kumar

Ananth Grama

Anshul Gupta

George Karypis

University of Minnesota



The Benjamin/Cummings Publishing Company, Inc.

Redwood City, California ■ Menlo Park, California

Reading, Massachusetts ■ New York ■ Don Mills, Ontario ■ Wokingham, U.K.

Amsterdam ■ Bonn ■ Sydney ■ Singapore ■ Tokyo ■ Madrid ■ San Juan

Executive editor: *Dan Joraanstad*
Sponsoring editor: *Carter Shanklin*
Editorial assistant: *Melissa Standen*
Production supervisor: *Gwen Larson*
Production management: *Matrix Productions*
Cover design: *Yvo Riezebos Design*

Copyright © 1994 by The Benjamin/Cummings Publishing Company, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Additional copyright and trademark information can be found at the end of the book. Printed in the United States of America. Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data

Introduction to parallel computing : design and analysis of parallel algorithms

Vipin Kumar... [et al.].

p. cm.

Includes biographical references and index.

ISBN 0-8053-3170-0

1. Parallel processing (Electronic computers) 2. Computer algorithms.

I. Grama, Ananth. II. Gupta, Anshul. III. Karypis, George.

QA76.58.I58 1994

005.2—dc20

93-34230

ISBN 0-8053-3170-0

4 5 6 7 8 9 10-CRW-98 97

The Benjamin/Cummings Publishing Company, Inc.

390 Bridge Parkway

Redwood City, California 94065

Preface

Parallel computers consisting of thousands of processors are now commercially available. These computers provide many orders of magnitude more raw computing power than traditional supercomputers at much lower cost. They open up new frontiers in the application of computers—many previously unsolvable problems can be solved if the power of these machines is used effectively. The availability of massively parallel computers has created a number of challenges, for example: How should parallel computers be programmed? What algorithms and data structures should be used? How can the quality of the algorithms be analyzed? Which algorithms are suitable for particular parallel computer architectures?

This book attempts to answer these and other questions about parallel computing. It presents a self-contained discussion of the basic concepts of parallel computer architectures and parallel algorithms for a variety of applications. The text is intended for senior undergraduate and graduate-level students, but is advanced enough to serve as a reference for practicing algorithm designers and application programmers. We hope that this book will bring an understanding of parallel processing to a wide range of people interested in solving problems on parallel computers.

Most of the material has been extensively class-tested. The book evolved out of a series of courses on related topics taught by the senior author, Vipin Kumar, over the past ten years. The other three authors have been actively conducting research on parallel computing under the supervision of the senior author. In Fall 1992, they joined him to prepare a comprehensive textbook on the design, analysis, and implementation of parallel algorithms on different parallel architectures. In particular, Ananth Grama took charge of Chapters 1, 2, 8, and 9; Anshul Gupta wrote Chapters 3, 4, 5, 10, and 11; and George Karypis was responsible for Chapters 6, 7, 12, and 13.

It is impossible to cover all the material in this book in a single course. However, a variety of courses can be taught using different chapters. Some suggestions for course contents are:

- (1) *Introduction to Parallel Computing*: Chapters 1 and 2, selected parts of Chapters 3, 4, and 13, and a sample of algorithms from the remaining chapters.
- (2) *Design and Analysis of Parallel Algorithms*: Chapter 1, portions of Chapters 2, 3, and 4, and selected algorithms from Chapters 5 through 12.

- (3) *Parallel Numerical Algorithms* or *Parallel Scientific Computing*: Chapters 1, 5, and 10–12, and parts of Chapters 2–4 and 13.

The senior author has been teaching a two-quarter sequence titled *Introduction to Parallel Computing* in the Computer Science Department at the University of Minnesota using drafts of the book. The first seven chapters are covered in the first quarter, and the remaining six in the second quarter. The senior author also teaches the course *High Performance Computing* for the scientific computing program at the University of Minnesota. This course is taken primarily by graduate students in the sciences and engineering (such as mechanical engineering, chemistry, and biology) who are interested in solving computationally intensive problems on parallel computers. This course covers parts of Chapters 1–5, 11, and 13.

Most chapters of the book include (1) examples and illustrations, (2) problems that supplement the text and test students' understanding of the material, and (3) bibliographic remarks to aid researchers and students interested in learning more about related and advanced topics. The notation used to express the complexity of functions and order analysis is explained in Appendix A. A glossary was originally planned, but was dropped in favor of a comprehensive index. In the index, the number of the page on which a term is explicitly defined appears in boldface type. Furthermore, the term itself appears in bold italics where it is defined. The sections that deal with relatively complicated material are preceded by a '★'. An instructors' manual containing slides of the figures and solutions to selected problems is also available from the publisher.

We view this book as a continually evolving resource. Readers are encouraged to send suggestions and information related to the material in the book to the authors, preferably by electronic mail to book-vk@cs.umn.edu. We welcome ideas, opinions, critiques, new problems, and programs for the algorithms in the book. Any such input will be added to the information archived in the directory bc/kumar at the anonymous FTP site bc.aw.com with due credits to the sender(s). On-line errata for the text will be maintained at the same site. In the highly-dynamic field of parallel computing, there is a lot to be gained from a healthy exchange of ideas in this manner.

Acknowledgements

Working on this project has been a source of great pleasure for us. At this juncture, we would like to acknowledge the people who worked with us and helped make this project a reality. We are most indebted to **Tom Nurkkala** and **Daniel Challou**, whose untiring editorial efforts have gone a long way to improve the quality of the text. They read every chapter several times, and gave technical, grammatical, and typesetting comments. Furthermore, their suggestions led to the addition of many new examples and illustrations. Without their contribution, this project would have stretched far longer and the impact of the book would have been diminished. We also express our gratitude to **Michael Heath**, who took great personal interest in the project. His comments were invaluable in improving both

the technical content and the quality of the material presented. We appreciate the efforts of Gregory Andrews, Daniel Boley, Shantanu Dutt, Rob Fowler, Joydeep Ghosh, Dirk Grunwald, John Gustafson, Charles Martel, Dan Miranker, Viktor Prasanna, Youcef Saad, and Vikram Saleetore in reviewing the manuscript and suggesting improvements. Victor Prasanna used parts of the preliminary drafts of the book for his class and provided valuable feedback. Any remaining errors or omissions in the text are the sole responsibility of the authors.

Many other people contributed to this project in different ways. We thank Jake Aggarwal, Gul Agha, Mani Chandy, Tom Cormen, Tse-Yun Feng, David Fox, Robert Hiromoto, Kai Hwang, Bharat Jairaman, L. V. Kale, Laveen Kanal, Tom Leighton, Babu Narayanan, Lionel Ni, Michael Quinn, Ben Rosen, Sartaj Sahni, Ahmed Sameh, Vineet Singh, Larry Snyder, and N. R. Vempaty for providing valuable input at various stages. We thank the students of the *Introduction to Parallel Computing* course at the University of Minnesota for identifying and working through errors in the drafts. In particular, comments from Minesh Amin, Dan Frankowski, Dave Truckenmiller, and Steve Waldo were very useful. We are thankful to Amy Gaukel for meticulously checking all the references and correcting the errors in them. It was a pleasure to work with the cooperative and helpful staff at Benjamin/Cummings. In particular, we thank John Carter Shanklin, Melissa Standen, Merrill Peterson, Dan Joranstad, and Adam Ray for their effort and cooperation.

We thank our family members, Akash, Chethan, Kalpana, Krista, Renu, Vipasha, and Anshu, for their affectionate support, patience, and encouragement throughout the duration of this project.

The Army Research Office (ARO) and the National Science Foundation provided support to the senior author for parallel computing research over the last decade. In particular, we thank Dr. Jag Chandra and Dr. Ken Clark of the ARO for supporting our research on scalable parallel algorithms at an early stage. Many of the results of this research appear in the book. The Army High Performance Computing Research Center and the Department of Computer Science at the University of Minnesota provided an active and nurturing environment for conducting research. We express our gratitude to the University of Minnesota for the computing facilities used to prepare the manuscript.

Vipin Kumar
Ananth Grama
Anshul Gupta
George Karypis

Minneapolis, Minnesota

Contents

Preface xiii

CHAPTER 1

Introduction 1

- 1.1 What is Parallel Computing? 2
- 1.2 The Scope of Parallel Computing 3
- 1.3 Issues in Parallel Computing 4
- 1.4 Organization and Contents of the Text 5
- 1.5 Bibliographic Remarks 8
 - Problems 9
 - References 10

CHAPTER 2

Models of Parallel Computers 15

- 2.1 A Taxonomy of Parallel Architectures 16
 - 2.1.1 *Control Mechanism* 16
 - 2.1.2 *Address-Space Organization* 18
 - 2.1.3 *Interconnection Networks* 22
 - 2.1.4 *Processor Granularity* 22
- 2.2 An Idealized Parallel Computer 23
- 2.3 Dynamic Interconnection Networks 24
 - 2.3.1 *Crossbar Switching Networks* 24
 - 2.3.2 *Bus-Based Networks* 25
 - 2.3.3 *Multistage Interconnection Networks* 26
- 2.4 Static Interconnection Networks 30
 - 2.4.1 *Types of Static Interconnection Networks* 30
 - 2.4.2 *Evaluating Static Interconnection Networks* 37

| | | |
|-------|--|----|
| 2.5 | Embedding Other Networks into a Hypercube | 39 |
| 2.5.1 | <i>Embedding a Linear Array into a Hypercube</i> | 39 |
| 2.5.2 | <i>Embedding a Mesh into a Hypercube</i> | 41 |
| 2.5.3 | <i>Embedding a Binary Tree into a Hypercube</i> | 42 |
| 2.6 | Routing Mechanisms for Static Networks | 42 |
| 2.7 | Communication Costs in Static Interconnection Networks | 45 |
| 2.7.1 | <i>Store-and-Forward Routing</i> | 45 |
| 2.7.2 | <i>Cut-Through Routing</i> | 46 |
| 2.8 | Cost-Performance Tradeoffs | 48 |
| 2.9 | Architectural Models for Parallel Algorithm Design | 49 |
| 2.10 | Bibliographic Remarks | 51 |
| | Problems | 54 |
| | References | 60 |

CHAPTER 3

| | | |
|---------------------------------------|--|-----------|
| Basic Communication Operations | | 65 |
| 3.1 | Simple Message Transfer between Two Processors | 66 |
| 3.2 | One-to-All Broadcast | 66 |
| 3.2.1 | <i>Store-and-Forward Routing</i> | 67 |
| 3.2.2 | <i>Cut-Through Routing</i> | 75 |
| 3.3 | All-to-All Broadcast, Reduction, and Prefix Sums | 77 |
| 3.3.1 | <i>Store-and-Forward Routing</i> | 78 |
| 3.3.2 | <i>Cut-Through Routing</i> | 86 |
| 3.4 | One-to-All Personalized Communication | 88 |
| 3.5 | All-to-All Personalized Communication | 90 |
| 3.5.1 | <i>Store-and-Forward Routing</i> | 90 |
| 3.5.2 | <i>Cut-Through Routing</i> | 95 |
| 3.6 | Circular Shift | 98 |
| 3.6.1 | <i>Store-and-Forward Routing</i> | 98 |
| 3.6.2 | <i>Cut-Through Routing</i> | 101 |
| 3.7 | Faster Methods for Some Communication Operations | 101 |
| 3.7.1 | <i>Routing Messages in Parts</i> | 102 |
| 3.7.2 | <i>All-Port Communication</i> | 104 |
| 3.7.3 | <i>Special Hardware for Global Operations</i> | 105 |
| 3.8 | Summary | 106 |
| 3.9 | Bibliographic Remarks | 107 |
| | Problems | 108 |
| | References | 114 |

CHAPTER 4**Performance and Scalability of Parallel Systems 117**

| | | |
|-------|--|-----|
| 4.1 | Performance Metrics for Parallel Systems | 117 |
| 4.1.1 | <i>Run Time</i> | 117 |
| 4.1.2 | <i>Speedup</i> | 118 |
| 4.1.3 | <i>Efficiency</i> | 120 |
| 4.1.4 | <i>Cost</i> | 120 |
| 4.2 | The Effect of Granularity and Data Mapping on Performance | 121 |
| 4.3 | The Scalability of Parallel Systems | 126 |
| 4.4 | The Isefficiency Metric of Scalability | 128 |
| 4.4.1 | <i>Problem Size</i> | 129 |
| 4.4.2 | <i>The Overhead Function</i> | 129 |
| 4.4.3 | <i>The Isefficiency Function</i> | 130 |
| 4.4.4 | <i>Cost-Optimality and the Isefficiency Function</i> | 133 |
| 4.4.5 | <i>A Lower Bound on the Isefficiency Function</i> | 134 |
| 4.4.6 | <i>The Degree of Concurrency and the Isefficiency Function</i> | 134 |
| 4.5 | Sources of Parallel Overhead | 135 |
| 4.5.1 | <i>Interprocessor Communication</i> | 135 |
| 4.5.2 | <i>Load Imbalance</i> | 135 |
| 4.5.3 | <i>Extra Computation</i> | 136 |
| 4.6 | Minimum Execution Time and Minimum Cost-Optimal Execution Time | 136 |
| 4.7 | Other Scalability Metrics and Bibliographic Remarks | 139 |
| | Problems | 141 |
| | References | 146 |

CHAPTER 5**Dense Matrix Algorithms 151**

| | | |
|-------|------------------------------------|-----|
| 5.1 | Mapping Matrices onto Processors | 151 |
| 5.1.1 | <i>Striped Partitioning</i> | 151 |
| 5.1.2 | <i>Checkerboard Partitioning</i> | 152 |
| 5.2 | Matrix Transposition | 153 |
| 5.2.1 | <i>Checkerboard Partitioning</i> | 154 |
| 5.2.2 | <i>Striped Partitioning</i> | 159 |
| 5.3 | Matrix-Vector Multiplication | 160 |
| 5.3.1 | <i>Rowwise Striping</i> | 160 |
| 5.3.2 | <i>Checkerboard Partitioning</i> | 163 |
| 5.4 | Matrix Multiplication | 168 |
| 5.4.1 | <i>A Simple Parallel Algorithm</i> | 169 |

| | | |
|-------|--|-----|
| 5.4.2 | <i>Cannon's Algorithm</i> | 171 |
| 5.4.3 | <i>Fox's Algorithm</i> | 173 |
| 5.4.4 | <i>The DNS Algorithm</i> | 174 |
| 5.5 | Solving a System of Linear Equations | 178 |
| 5.5.1 | <i>A Simple Gaussian Elimination Algorithm</i> | 179 |
| 5.5.2 | <i>Gaussian Elimination with Partial Pivoting</i> | 192 |
| 5.5.3 | <i>Solving a Triangular System: Back-Substitution</i> | 195 |
| 5.5.4 | <i>Numerical Considerations in Solving Systems of Linear Equations</i> | 196 |
| 5.6 | Bibliographic Remarks | 197 |
| | Problems | 198 |
| | References | 204 |

CHAPTER 6

Sorting 209

| | | |
|-------|---|-----|
| 6.1 | Issues in Sorting on Parallel Computers | 210 |
| 6.1.1 | <i>Where the Input and Output Sequences Are Stored</i> | 210 |
| 6.1.2 | <i>How Comparisons Are Performed</i> | 210 |
| 6.2 | Sorting Networks | 212 |
| 6.2.1 | <i>Bitonic Sort</i> | 214 |
| 6.2.2 | <i>Mapping Bitonic Sort onto a Hypercube and a Mesh</i> | 216 |
| 6.3 | Bubble Sort and Its Variants | 224 |
| 6.3.1 | <i>Odd-Even Transposition</i> | 225 |
| 6.3.2 | <i>Shellsort</i> | 227 |
| 6.4 | Quicksort | 229 |
| 6.4.1 | <i>Parallelizing Quicksort</i> | 231 |
| 6.4.2 | <i>A Comparison of Quicksort Formulations</i> | 243 |
| 6.5 | Other Sorting Algorithms | 243 |
| 6.5.1 | <i>Enumeration Sort</i> | 243 |
| 6.5.2 | <i>Bucket Sort</i> | 244 |
| 6.5.3 | <i>Sample Sort</i> | 245 |
| 6.5.4 | <i>Radix Sort</i> | 247 |
| 6.6 | Bibliographic Remarks | 247 |
| | Problems | 250 |
| | References | 254 |

CHAPTER 7

Graph Algorithms 257

| | | |
|-----|--------------------------------|-----|
| 7.1 | Definitions and Representation | 257 |
|-----|--------------------------------|-----|

| | | |
|-------|--|-----|
| 7.2 | Minimum Spanning Tree: Prim's Algorithm | 260 |
| 7.3 | Single-Source Shortest Paths: Dijkstra's Algorithm | 265 |
| 7.4 | All-Pairs Shortest Paths | 266 |
| 7.4.1 | <i>Matrix-Multiplication Based Algorithm</i> | 266 |
| 7.4.2 | <i>Dijkstra's Algorithm</i> | 268 |
| 7.4.3 | <i>Floyd's Algorithm</i> | 271 |
| 7.4.4 | <i>Performance Comparisons</i> | 276 |
| 7.5 | Transitive Closure | 276 |
| 7.6 | Connected Components | 278 |
| 7.6.1 | <i>A Depth-First Search Based Algorithm</i> | 278 |
| 7.7 | Algorithms for Sparse Graphs | 281 |
| 7.7.1 | <i>Single-Source Shortest Paths</i> | 284 |
| 7.8 | Bibliographic Remarks | 290 |
| | Problems | 293 |
| | References | 295 |

CHAPTER 8

Search Algorithms for Discrete Optimization Problems 299

| | | |
|-------|---|-----|
| 8.1 | Definitions and Examples | 299 |
| 8.2 | Sequential Search Algorithms | 304 |
| 8.2.1 | <i>Depth-First Search Algorithms</i> | 304 |
| 8.2.2 | <i>Best-First Search Algorithms</i> | 308 |
| 8.3 | Search Overhead Factor | 308 |
| 8.4 | Parallel Depth-First Search | 310 |
| 8.4.1 | <i>Important Parameters of Parallel DFS</i> | 313 |
| 8.4.2 | <i>A General Framework for Analysis of Parallel DFS</i> | 315 |
| 8.4.3 | <i>Analysis of Load-Balancing Schemes for Hypercubes</i> | 318 |
| 8.4.4 | <i>Analysis of Load-Balancing Schemes for a Network of Workstations</i> | 320 |
| 8.4.5 | <i>Termination Detection</i> | 321 |
| 8.4.6 | <i>Experimental Results</i> | 325 |
| 8.4.7 | <i>Parallel Formulations of Depth-First Branch-and-Bound Search</i> | 328 |
| 8.4.8 | <i>Parallel Formulations of IDA*</i> | 328 |
| 8.4.9 | <i>Parallel DFS on SIMD Computers</i> | 329 |
| 8.5 | Parallel Best-First Search | 332 |
| 8.6 | Speedup Anomalies in Parallel Search Algorithms | 336 |
| 8.6.1 | <i>Analysis of Average Speedup in Parallel DFS</i> | 338 |
| 8.7 | Bibliographic Remarks | 340 |
| | Problems | 343 |
| | References | 348 |

CHAPTER 9

Dynamic Programming 355

- 9.1 Serial Monadic DP Formulations 357
 - 9.1.1 *The Shortest-Path Problem* 358
 - 9.1.2 *The 0/1 Knapsack Problem* 360
- 9.2 Nonserial Monadic DP Formulations 362
 - 9.2.1 *The Longest-Common-Subsequence Problem* 363
- 9.3 Serial Polyadic DP Formulations 365
 - 9.3.1 *Floyd's All-Pairs Shortest-Paths Algorithm* 365
- 9.4 Nonserial Polyadic DP Formulations 366
 - 9.4.1 *The Optimal Matrix-Parentesization Problem* 366
- 9.5 Summary and Discussion 369
- 9.6 Bibliographic Remarks 370
 - Problems 371
 - References 375

CHAPTER 10

Fast Fourier Transform 377

- 10.1 The Serial Algorithm 377
- 10.2 The Binary-Exchange Algorithm 382
 - 10.2.1 *Hypercube* 383
 - 10.2.2 *Mesh* 388
 - 10.2.3 *Extra Computations in Parallel FFT* 390
- 10.3 The Transpose Algorithm 393
 - 10.3.1 *Two-Dimensional Transpose Algorithm* 393
 - 10.3.2 *The Generalized Transpose Algorithm* 396
- 10.4 Cost-Effectiveness of Meshes and Hypercubes for FFT 400
- 10.5 Bibliographic Remarks 403
 - Problems 404
 - References 405

CHAPTER 11

Solving Sparse Systems of Linear Equations 407

- 11.1 Basic Operations 409
 - 11.1.1 *Storage Schemes for Sparse Matrices* 409
 - 11.1.2 *Vector Inner Product* 412

| | | |
|--------|---|-----|
| 11.1.3 | <i>Sparse Matrix-Vector Multiplication</i> | 413 |
| 11.2 | Iterative Methods for Sparse Linear Systems | 426 |
| 11.2.1 | <i>Jacobi Iterative Method</i> | 427 |
| 11.2.2 | <i>Gauss-Seidel and SOR Methods</i> | 429 |
| 11.2.3 | <i>The Conjugate Gradient Method</i> | 433 |
| 11.3 | Finite Element Method | 446 |
| 11.4 | Direct Methods for Sparse Linear Systems | 454 |
| 11.4.1 | <i>Ordering</i> | 455 |
| 11.4.2 | <i>Symbolic Factorization</i> | 458 |
| 11.4.3 | <i>Numerical Factorization</i> | 458 |
| 11.4.4 | <i>Solving a Triangular System</i> | 468 |
| 11.5 | Multigrid Methods | 468 |
| 11.6 | Bibliographic Remarks | 474 |
| | Problems | 477 |
| | References | 482 |

CHAPTER 12

Systolic Algorithms and their Mapping onto Parallel Computers 491

| | | |
|--------|---|-----|
| 12.1 | Examples of Systolic Systems | 493 |
| 12.1.1 | <i>Convolution</i> | 493 |
| 12.1.2 | <i>Banded Matrix-Vector Multiplication</i> | 496 |
| 12.1.3 | <i>Matrix Multiplication</i> | 499 |
| 12.1.4 | <i>Optimal Matrix Parenthesization</i> | 501 |
| 12.2 | General Issues in Mapping Systolic Systems onto Parallel Computers | 505 |
| 12.2.1 | <i>Architectural Differences between Systolic Arrays and Parallel Computers</i> | 505 |
| 12.2.2 | <i>Absolute Efficiency of Systolic Systems</i> | 506 |
| 12.3 | Mapping One-Dimensional Systolic Arrays | 507 |
| 12.3.1 | <i>Virtual Processors</i> | 508 |
| 12.3.2 | <i>Block-Striped Mapping</i> | 508 |
| 12.3.3 | <i>Other One-Dimensional Mappings</i> | 511 |
| 12.4 | Mapping Two-Dimensional Systolic Arrays | 513 |
| 12.4.1 | <i>Block-Checkerboard Mapping</i> | 514 |
| 12.4.2 | <i>Cyclic-Checkerboard Mapping</i> | 516 |
| 12.4.3 | <i>Summary of Two-Dimensional Mappings</i> | 518 |
| 12.5 | Bibliographic Remarks | 518 |
| | Problems | 520 |
| | References | 521 |

CHAPTER 13

Parallel Programming 525

- 13.1 Parallel Programming Paradigms 525
 - 13.1.1 *Explicit versus Implicit Parallel Programming* 525
 - 13.1.2 *Shared-Address-Space versus Message-Passing* 526
 - 13.1.3 *Data Parallelism versus Control Parallelism* 527
- 13.2 Primitives for the Message-Passing Programming Paradigm 529
 - 13.2.1 *Basic Extensions* 529
 - 13.2.2 *nCUBE 2* 531
 - 13.2.3 *iPSC 860* 532
 - 13.2.4 *CM-5* 533
 - 13.2.5 *Example Program* 535
- 13.3 Data-Parallel Languages 538
 - 13.3.1 *Data Partitioning and Virtual Processors* 539
 - 13.3.2 *C** 540
 - 13.3.3 *CM Fortran* 549
- 13.4 Primitives for the Shared-Address-Space Programming Paradigm 551
 - 13.4.1 *Primitives to Allocate Shared Variables* 552
 - 13.4.2 *Primitives for Mutual Exclusion and Synchronization* 552
 - 13.4.3 *Primitives for Creating Processes* 553
 - 13.4.4 *Sequent Symmetry* 553
 - 13.4.5 *Example Programs* 554
- 13.5 Fortran D 558
 - 13.5.1 *Problem Mapping* 559
 - 13.5.2 *Machine Mapping* 561
 - 13.5.3 *Example Program* 562
- 13.6 Bibliographic Remarks 564
 - References 566

APPENDIX A

Complexity of Functions and Order Analysis 571

Author Index 575

Subject Index 583

Introduction

Ever since conventional serial computers were invented, their speed has steadily increased to match the needs of emerging applications. However, the fundamental physical limitation imposed by the speed of light makes it impossible to achieve further improvements in the speed of such computers indefinitely. Recent trends show that the performance of these computers is beginning to saturate. A natural way to circumvent this saturation is to use an ensemble of processors to solve problems.

A cost-performance comparison of serial computers over the last few decades shows an interesting evolutionary trend. Figure 1.1 represents typical cost-performance curves of serial computers over the past three decades. At the lower end of each curve, performance increases almost linearly (or even faster than linearly) with cost. However, beyond a certain point, each curve starts to saturate, and even small gains in performance come at an exorbitant increase in cost. Furthermore, this transition point has become sharper with the passage of time, primarily as a result of advances in very large scale integration (VLSI)

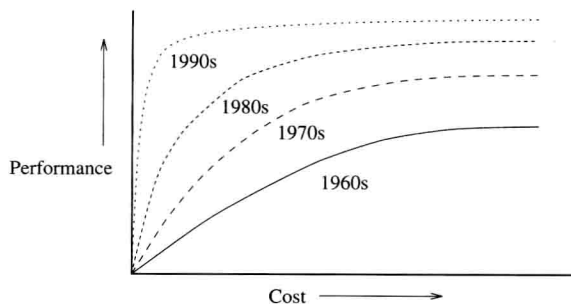


Figure 1.1 Cost versus performance curve and its evolution over the decades.

technology. It is now possible to construct very fast, low-cost processors. This increases the demand for and production of these processors, resulting in lower prices.

Currently, the speed of off-the-shelf microprocessors is within one order of magnitude of the speed of the fastest serial computers. However, microprocessors cost many orders of magnitude less. This implies that, by connecting only a few microprocessors together to form a parallel computer, it is possible to obtain raw computing power comparable to that of the fastest serial computers. Typically, the cost of such a parallel computer is considerably less.

Furthermore, connecting a large number of processors into a parallel computer overcomes the saturation point of the computation rates achievable by serial computers. Thus, parallel computers can provide much higher raw computation rates than the fastest serial computers as long as this power can be translated into high computation rates for actual applications.

1.1 What is Parallel Computing?

This section illustrates some important aspects of parallel computing by drawing an analogy to a real-life scenario.

Consider the problem of stacking (reshelving) a set of library books. A single worker trying to stack all the books in their proper places cannot accomplish the task faster than a certain rate. We can speed up this process, however, by employing more than one worker. Assume that the books are organized into shelves and that the shelves are grouped into bays. One simple way to assign the task to the workers is to divide the books equally among them. Each worker stacks the books one at a time. This division of work may not be the most efficient way to accomplish the task, since the workers must walk all over the library to stack books. An alternate way to divide the work is to assign a fixed and disjoint set of bays to each worker. As before, each worker is assigned an equal number of books arbitrarily. If a worker finds a book that belongs to a bay assigned to him or her, he or she places that book in its assigned spot. Otherwise, he or she passes it on to the worker responsible for the bay it belongs to. The second approach requires less effort from individual workers.

The preceding example shows how a task can be accomplished faster by dividing it into a set of subtasks assigned to multiple workers. Workers cooperate, pass the books to each other when necessary, and accomplish the task in unison. Parallel processing works on precisely the same principles. Dividing a task among workers by assigning them a set of books is an instance of **task partitioning**. Passing books to each other is an example of **communication** between subtasks.

Problems are parallelizable to different degrees. For some problems, assigning partitions to other processors might be more time-consuming than performing the processing locally. Other problems may be completely serial. For example, consider the task of digging a post hole. Although one person can dig a hole in a certain amount of time, employing more people does not reduce this time. Because it is impossible to partition this

task, it is poorly suited to parallel processing. Therefore, a problem may have different parallel formulations, which result in varying benefits, and all problems are not equally amenable to parallel processing.

1.2 The Scope of Parallel Computing

Parallel processing is making a tremendous impact on many areas of computer application. With the high raw computing power of parallel computers, it is now possible to address many applications that were until recently beyond the capability of conventional computing techniques.

Many applications, such as weather prediction, biosphere modeling, and pollution monitoring, are modeled by imposing a grid over the domain being modeled. The entities within grid elements are simulated with respect to the influence of other entities and their surroundings. In many cases, this requires solutions to large systems of differential equations. The granularity of the grid determines the accuracy of the model. Since many such systems are evolving with time, time forms an additional dimension for these computations. Even for a small number of grid points, a three-dimensional coordinate system, and a reasonable discretized time step, this modeling process can involve trillions of operations (Example 1.1). Thus, even moderate-sized instances of these problems take an unacceptably long time to solve on serial computers.

Example 1.1 Weather Modeling and Forecasting

Consider the modeling of weather over an area of 3000×3000 miles. The parameters must also be modeled along the vertical plane. Assume that the area is being modeled up to a height of 11 miles. Assume that the $3000 \times 3000 \times 11$ cubic mile domain is partitioned into segments of size $0.1 \times 0.1 \times 0.1$ cubic miles. There are approximately 10^{11} different segments. The weather modeling process involves time as another dimension. Time is quantized and parameters are computed for each segment at regular time intervals.

Let us further assume that we are modeling the weather over a two-day period and the parameters need to be computed once every half hour. (Note that the assumptions are conservative and more accurate modeling requires much higher computation rates.) The computation of parameters inside a segment uses the initial values and the values from neighboring segments. Assume that this computation takes 100 instructions. Therefore, a single updating of the parameters in the entire domain requires $10^{11} \times 100$, or 10^{13} instructions. Since this has to be done approximately 100 times (two days), the total number of operations is 10^{15} . On a serial supercomputer capable of performing one billion instructions per second, this modeling would take approximately 280 hours. Taking 280 hours to predict the weather for the next 48 hours is unreasonable to say the least.

Parallel processing makes it possible to predict the weather not only faster but also more accurately. If we have a parallel computer with a thousand workstation-