

Fundamentals of Parallel Processing 并行处理基础

Harry F. Jordan
Gita Alaghband

This unique book provides comprehensive coverage of the crucial fundamentals of parallel processing. The authors also introduce the integration of parallel architectures, algorithms, and language to provide insight into designing and implementing applications. This book is suitable for advanced courses on the principles of parallel processing and is also a superb professional reference.

Beneficial Features:

- ▶ Presents subjects by integrating algorithms, languages, and architectures.
- ▶ Clearly explains performance analysis and implications resulting from influencing three components.
- ▶ Guides readers with thought-provoking questions at the beginning of each chapter.
- ▶ Offers numerous examples to clarify difficult concepts.
- ▶ Helps students implement their learning with 141 end-of-chapter problems.
- ▶ Conveys performance implications of each new topic with consistent integration of algorithms, languages, and architectures throughout the text.

About the Authors

Harry F. Jordan received the Ph.D. from the University of Illinois. He has been at the University of Colorado at Boulder since 1966 and is now a professor in the Department of Electrical and Computer Engineering and Computer Science. Professor Jordan's research interests include computer systems center on the interface between hardware and software, including supercomputers, multi-processor architecture, and optical computing.

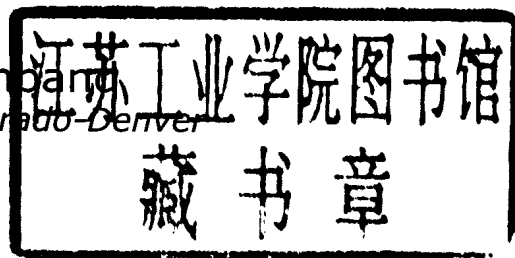
Gita Alaghband received the Ph.D. in Electrical Engineering (1986) from the University of Colorado at Boulder. Currently, she is a professor in the Department of Computer Engineering at the University of Colorado at Denver. Dr. Alaghband's research interests include parallel processing including computer architecture, performance evaluation, simulation, application programs, and algorithm designs.

**Fundamentals of
Parallel Processing**

并行处理基础

Harry F. Jordan
University of Colorado-Boulder

Gita Alaghband
University of Colorado-Denver



清华大学出版社
北京

English reprint edition copyright © 2003 by **PEARSON EDUCATION ASIA LIMITED and TSINGHUA UNIVERSITY PRESS.**

Original English language title from Proprietor's edition of the Work.

Original English language title: Fundamentals of Parallel Processing, Harry F. Jordan, Gita Alaghband, Copyright © 2003

All Rights Reserved.

Published by arrangement with the original publisher, Pearson Education, Inc., publishing as Prentice Hall, Inc.

This edition is authorized for sale and distribution only in the People's Republic of China (excluding the Special Administrative Region of Hong Kong, Macao SAR and Taiwan).

本书影印版由 Prentice Hall, Inc. 授权给清华大学出版社出版发行。

**For sale and distribution in the People's Republic of China exclusively
(except Taiwan, Hong Kong SAR and Macao SAR).**

**仅限于中华人民共和国境内(不包括中国香港、澳门特别行政区和
中国台湾地区)销售发行。**

北京市版权局著作权合同登记号 图字: 01-2003-3083

本书封面贴有 **Pearson Education** (培生教育出版集团) 激光防伪标签, 无标签者不得销售。

图书在版编目(CIP)数据

并行处理基础 = Fundamentals of Parallel Processing / [美] 乔丹 (Jordan, H.), 阿拉盖班德 (Alaghband, G.) 著. — 影印本. — 北京: 清华大学出版社, 2003

(计算机科学与技术学科研究生系列教材)

ISBN 7-302-07382-1

I. 并… II. ①乔… ②阿… III. 并行处理—研究生—教材—英文 IV. TP274

中国版本图书馆 CIP 数据核字 (2003) 第 091033 号

出 版 者: 清华大学出版社

<http://www.tup.com.cn>

社总机: (010) 6277 0175

责任编辑: 马瑛琨

印 刷 者: 北京密云胶印厂

发 行 者: 新华书店总店北京发行所

开 本: 185×230 印张: 34.75 字数: 699 千字

版 次: 2003 年 10 月第 1 版 2003 年 10 月第 1 次印刷

书 号: ISBN 7-302-07382-1/TP·5357

印 数: 1~5000

定 价: 56.00 元

地 址: 北京清华大学学研大厦

邮 编: 100084

客户服务: (010) 6277 6969

封面设计: 孟繁聪

装 订 者: 北京鑫海金澳胶印有限公司

编
委
会

■ 名誉主任：陈火旺

■ 主 任：王志英

■ 副 主 任：钱德沛 周立柱

■ 编委委员：（按姓氏笔画为序）

马殿富 李晓明 李仲麟 吴朝晖

何炎祥 陈道蓄 周兴社 钱乐秋

蒋宗礼 焦金生 廖明宏

■ 责任编辑：马瑛珺

本书责任编辑：王志英

序

未来的社会是信息化的社会,计算机科学与技术在其中占据了最重要的地位,这对高素质创新型计算机人才的培养提出了迫切的要求。计算机科学与技术已经成为一门基础技术学科,理论性和技术性都很强。与传统的数学、物理和化学等基础学科相比,该学科的教育工作者既要培养学科理论研究和基本系统的开发人才,还要培养应用系统开发人才,甚至是应用人才。从层次上来讲,则需要培养系统的设计、实现、使用与维护等各个层次的人才。这就要求我们的计算机教育按照定位的需要,从知识、能力、素质三个方面进行人才培养。

硕士研究生的教育需突出“研究”,要加强理论基础的教育和科研能力的训练,使学生能够站在一定的高度去分析研究问题、解决问题。硕士研究生要通过课程的学习,进一步提高理论水平,为今后的研究和发展打下坚实的基础;通过相应的研究及学位论文撰写工作来接受全面的科研训练,了解科学研究的艰辛和科研工作者的奉献精神,培养良好的科研作风,锻炼攻关能力,养成协作精神。

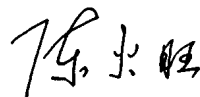
高素质创新型计算机人才应具有较强的实践能力,教学与科研相结合是培养实践能力的有效途径。高水平人才的培养是通过被培养者的高水平学术成果来反映的,而高水平的学术成果主要来源于大量高水平的科研。高水平的科研还为教学活动提供了最先进的高新技术平台和创造性的工作环境,使学生得以接触最先进的计算机理论、技术和环境。高水平的科研也为高水平人才的素质教育提供了良好的物质基础。

为提高高等院校的教学质量,教育部最近实施了精品课程建设工程。由于教材是提高教学质量的关键,必须加快教材建设的步伐。为适应学科的快速发展和培养方案的需要,要采取多种措施鼓励从事前沿研究的学者参与教材的编写和更新,在教材中反映学科前沿的研究成果与发展趋势,以高水平的科研促进教材建设。同时应适当引进国外先进的原版教材,确保所有教学环节充分反映计算机学科与产业的前沿研究水平,并与未来的发展趋势相协调。

中国计算机学会教育专业委员会在清华大学出版社的大力支持下,进行了计算机科学与技术学科硕士研究生培养的系统研究。在此基础上组织来自多所全国重点大学的计算机专家和教授们编写和出版了本系列教材。作者们以自己多年来丰富的教学和科研经验为基础,认真研究和结合我国计算机科学与技术学科硕士研究生教育的特点,力图使本系列教材对我国计算机科学与技术学科硕士研究生的教学方法和教学内容的改革起到引导作用。本系列教材的系统性和理论性强,学术水平高,反映科技新发展,具有合

适的深度和广度。同时本系列教材两种语种（中文、英文）并存，三种版权（本版、外版、合作出版）形式并存，这在系列教材的出版上走出了一条新路。

相信本系列教材的出版，能够对提高我国计算机硕士研究生教材的整体水平，进而对我国大学的计算机科学与技术硕士研究生教育以及培养高素质创新型计算机人才产生积极的促进作用。



2003 年 9 月

Preface

To the Student

Computing is usually taught from a step-by-step or serial point of view. Algorithms are organized as a sequence of computational steps, programs are written one command after another, and machines are designed to execute a chain of machine instructions by performing a string of microsteps, one after another. While sequential formulation of a problem can lead to a solution, a tremendous performance advantage is available from doing many operations in parallel. The two principal approaches to speeding up a computation are a faster clock rate for the underlying hardware and doing more operations in parallel. Introducing parallel operations to speed up an application is a promising approach, because as tasks become larger, more operations can potentially be done in parallel. To realize this potential, three things must work together. Algorithms must involve many independent operations, programming languages must allow the specification of parallel operations or identify them automatically, and the architecture of the computer running the program must execute multiple operations simultaneously.

Parallel processing is the result of this combination of algorithm design, programming language structure, and computer architecture all directed toward faster completion of an application. The fundamentals of parallel processing emerge from an understanding of this combination of computing topics and their collaboration to achieve high performance. To gain this understanding, a basic knowledge of computer design and architecture, of programming languages and how they produce machine code, and of the elements of algorithm structure is required. Although some subsections focus exclusively on one of the three aspects of architecture, language, or algorithm, there are no such major divisions in the book. Treatments of all three are combined to expose the fundamental concepts that make up the discipline of parallel processing. We expect the reader to have a basic knowledge of algorithms and programming. To

address the real goal of parallel processing—better performance—one must know how the program is executed by a computer at the machine language level. This requires an understanding of the specific organization of hardware elements constituting a machine architecture. Introductory experience in these areas constitutes the prerequisite material for reading this text.

To the Instructor

The goal of this textbook is to provide a comprehensive coverage of the principles of parallel processing. Integration of parallel architectures, algorithms, and languages is the key in gaining both the breadth and the depth of knowledge and expertise needed in designing and developing successful parallel applications. The book is organized and presented so that it continuously relates these subjects within the topic being studied. Discussions of algorithm designs are followed by the performance implications of each design on parallel architectures.

The rapid changes in technology and the continuous arrival of new architectures, languages, and systems demand a fundamental understanding of the field of parallel processing. The uniqueness of this book is that it treats fundamental concepts rather than a collection of the latest trends. The flow of information is carefully designed so that each section is a natural next step from the previous one. Detailed examples are used to clarify difficult concepts. The issues to be studied are posed early enough to motivate the reader to continue and to give a clear picture of what is to come next and why. The alternative approach of covering “recent” architectures, languages, and systems as a vehicle to teach the fundamental concepts is difficult and quickly dated. It is very hard to get to the heart of a subject without the readers feeling lost and confused about what is really being conveyed. Peeling off some layers of additional information and features is necessary before getting to the fundamentals in every case. For example, is it necessary for a language to provide numerous constructs? Or are some of them considered essential and some additionally provided for ease of use? Are they implemented with efficiency in mind for certain architectures or are they provided for portability? Are the constructs implementation dependent? Will their performance vary by much on different computer architectures? It is never possible to completely understand the trade-offs and the underlying concepts by going over example machines and languages alone. Once the fundamental concepts are understood, they can be applied to any architecture, system, or language.

Parallel processing is a relatively young academic discipline. The authors believe that it has developed to a point where fundamentals can be identified and discussed apart from individual systems. We have focused on presenting the fundamentals by architectural features, system properties, language constructs, algorithm design and implementation implications in a way that is as independent as possible of specific architectures, systems, and languages. In some cases, the original machine, language, or system introducing the concept being presented is covered. However, in a majority of cases we have intentionally refrained from expanding each topic to cover many specific machines or languages for the purpose of concentrating on fundamentals.

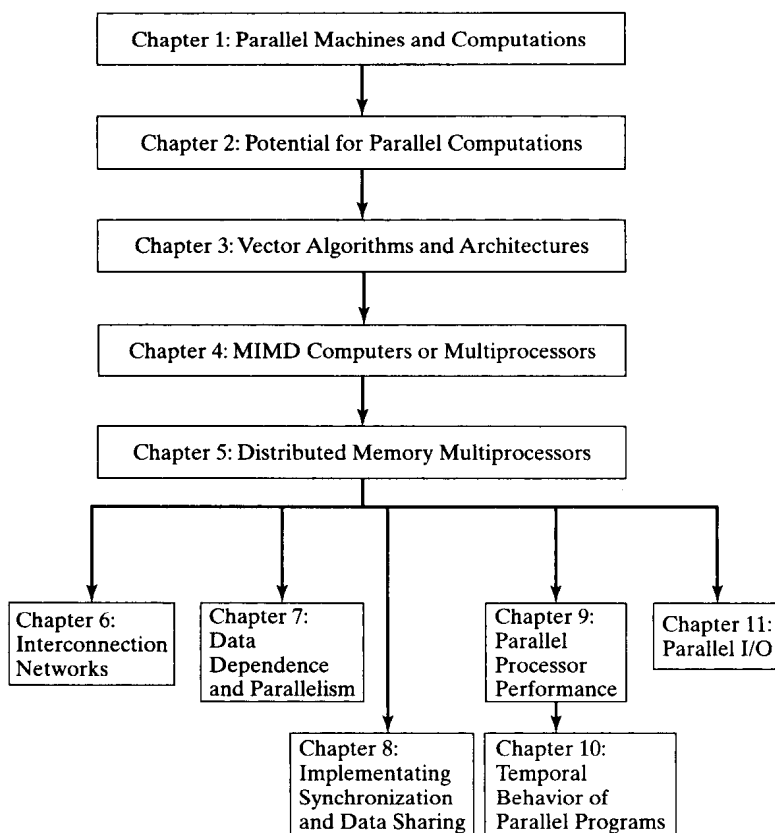
Although this is not intended as a parallel programming text, a real programming language is presented for each type of major parallelism concept introduced throughout the text. We selected Fortran as the base language whenever possible for several reasons. Much of the research literature in parallel processing is Fortran-based, and there are numerous parallel Fortran scientific programs and programmers. In addition, Fortran is a simple high-level language close to the machine level. It is easier to observe and explain the effects of executing Fortran statements on various machine architectures compared to high-level languages with many complex, user-friendly features. The Fortran program designer has much control over programming style, design, implementation, and execution. Fortran is a static language, so in comparison to dynamic languages or languages providing dynamic features, the programmer must be cautioned less regarding the use of high-level features and their parallel performance implications. The simplicity of the language helps keep the focus on parallel concepts and constructs. That multidimensional arrays are supported in Fortran is especially significant for vector processing. Maintaining the same base language throughout the book keeps the presentation consistent, and readers, not needing to switch between languages, will concentrate on parallel issues.

This textbook is designed and organized after many years of teaching and research experience in the field of parallel processing. It is intended for computer science or computer engineering seniors and graduate students. Students studying the book will be able to confidently design and implement new parallel applications, evaluate parallel program and architecture performance, and, most important be able to develop their skills by learning new parallel environments on their own. The major task of an educator is to nurture his or her students so that they can continue to grow and develop in their field of interest independently. This textbook is designed with this important goal in mind and will provide instructors with a comprehensive set of material to educate their students to be productive and successful.

Tailoring the Text to a Syllabus

The first seven chapters constitute an excellent first-semester course in parallel processing. They give an in-depth coverage of parallel algorithm design, vector, multiprocessor, and dataflow architectures, parallel languages for each machine type, synchronization and communication mechanisms, interconnection networks, data dependence, and compiler optimization techniques. The remaining four chapters are intended for advanced treatment of issues studied in the first part of the book. The focus is on various synchronization and communication implementations, influence of implementations on performance, interpretation of machine architecture and program performance, effects of program behavior on performance, and parallel I/O. This part of the book provides an excellent second semester for graduate students. They will gain insight into how to analyze machine architectures, parallel programs and systems, and understand how these components interact and influence overall performance. Many advanced research project ideas can be deduced from topics covered in Chapters 8 through 11.

The following chart shows the major dependencies among the chapters and suggests how they may be tailored to cover specific areas of emphasis.



Chapter Contents

Chapter 1 briefly reviews the evolution of parallelism in computer architectures. It introduces the basic ideas of vector processing, multiprocessing, and parallel operations in algorithms. It establishes a framework for topics in the remaining chapters.

Chapter 2 introduces the key ideas of data dependence. The prefix computation is used to illustrate algorithm characteristics that make different ways of doing the same computation more or less parallel.

Chapter 3 examines the application of the same operation to multiple data items in parallel. It motivates the discussion with some simple algorithms from linear algebra and presents an architecture at the machine language level that incorporates vector operations. Fortran 90 is

discussed as a language with high-level support for the unique features of machine level vector processing. Pipelined vector processing is discussed.

Chapter 4 briefly surveys multiprocessor architectural organizations and establishes the difference between shared and distributed memory multiprocessors. It proceeds to focus on shared memory by describing the extensions to sequential programming that are needed to coordinate multiple processes to perform a task. The OpenMP Fortran extension is used as an illustration of high-level constructs used to support shared memory multiprocessing. The chapter also establishes the basics of pipelined MIMD, or multithreaded, architectures.

Chapter 5 describes distributed memory multiprocessors using the message passing viewpoint to direct attention toward the dominant role of data communication in such architectures. Explicit send and receive programming is introduced, and the message passing interface (MPI) is used as an illustration of high-level language support for such programs. The basics of cache coherence and memory consistency are described in relation to shared memory and distributed memory multiprocessors.

Chapter 6 discusses interconnection networks in depth, including those for vector computers, and shared and distributed memory multiprocessors. Static and dynamic networks are compared and contrasted along with various topologies and their properties. Use of the network to combine messages, as in the NYU Ultracomputer, is discussed.

Chapter 7 is important in relating the ideas of data dependence that underlie the structure of parallel algorithms to the structure of a program. It covers code optimization techniques and topics of concern to a compiler writer having the task of generating code for a parallel computer. This chapter also introduces the ideas of dataflow languages and architectures that allow the elimination of nonessential dependences from programming languages and machines.

Chapter 8 expands the ideas of synchronization introduced in the shared memory discussion of Chapter 4 and integrates them with the data transmission point of view emphasized in Chapter 5. An in-depth understanding of key issues in synchronization is provided by a set of key topics, ranging from synchronization in cooperative communication, managing shared tasks, waiting mechanisms, to how to prove that a synchronization mechanism is implemented correctly.

Chapter 9 focuses specifically on the performance issues that have been continually referred to in previous chapters. It treats various performance models and illustrates their use through case studies of measurements on real systems. The impact of different scheduling and implementations of parallel constructs is discussed.

Chapter 10 relates performance of a parallel program execution to its temporal behavior. Experiments on real systems are used to illustrate performance characterization models. It examines temporal characterization from several viewpoints ranging from behavior in single cache systems, multiprocessor systems with distributed caches, to message passing systems.

Chapter 11 treats various aspects of parallelism in I/O operations. Parallel access disk arrays (RAID) are described as parallel I/O hardware. I/O dependence operations are introduced. Parallel input and output methods on files are discussed. Finally, parallelism in multiprocessors collective I/O operations is covered using MPI-IO.

Acknowledgments

Grateful acknowledgment is due to the Institute for Computer Applications in Science and Engineering, ICASE, which brought together a very stimulating group of computer scientists during the late 1970s and the 1980s. Considerable intellectual inspiration was derived from working with them. An important source of thoughts about the fundamentals of parallel processing came from the Conferences on High Speed Computing organized by Bill Buzbee and George Michael in 1980 and shepherded by them for 10 years. The effective mix of academics and application scientists they gathered at these meetings did much to establish the fundamentals of the field. Burton Smith has been a constant source of insight and perceptiveness. No technical conversation with him is ever very far from the fundamentals. Iain Duff at CERFACS, Centre European de Recherche et de Formation Avancee en Calcul Scientifique, provided a stimulating environment for collaboration with European scientists. We are grateful to Alan Apt, editor, and Prentice Hall for their invaluable support in completing this book.

The authors express their sincere appreciation to Chris Nevison, Wirg Wallentine, Pearl Wang, Tanya Zlateeva, Nan Schaller, David Kincaid, Norm Troullier, Steve Seidel, Hank Dietz, Richard Hughey, and Kathy Liszka for their diligent review of the book and constructive comments.

A very special thanks and deepest gratitude from Gita Alaghband to Harry Jordan to whom she is indebted for his years of generous guidance, mentorship, friendship, and support.

Finally, we express our appreciation to our families, Sue Jordan, Hamid, Sati, and Sara Fardi, for their loving support, patience, and encouragement throughout the years.

Contents

Preface	vii
Chapter 1: Parallel Machines and Computations	1
1.1 The Evolution of Parallel Architectures	2
1.1.1 Parallelism in Sequential Computers	2
1.1.2 Vector or SIMD Computers	6
1.1.3 Multiprocessors or MIMD Computers	8
1.2 Interconnection Networks	11
1.3 Application of Architectural Parallelism	12
1.4 Getting Started in SIMD and MIMD Programming	13
1.5 Parallelism in Algorithms	16
1.6 Conclusion	19
1.7 Bibliographic Notes	19
Chapter 2: Potential for Parallel Computations	23
2.1 Parameters Characterizing Algorithm Parallelism	23
2.2 Prefix Problem	25
2.3 Parallel Prefix Algorithms	26
2.3.1 Upper/Lower Parallel Prefix Algorithm	27
2.3.2 Odd/Even Parallel Prefix Algorithm	29
2.3.3 Ladner and Fischer's Parallel Prefix	31
2.4 Characterizing Algorithm Behavior for Large Problem Size	34
2.5 Programming Parallel Prefix	35

2.6	Speedup and Efficiency of Parallel Algorithms	36
2.7	The Performance Perspective	41
2.7.1	Factors That Influence Performance	41
2.7.2	A Simple Performance Model—Amdahl's Law	43
2.7.3	Average Execution Rate	45
2.8	Conclusion	46
2.9	Bibliographic Notes	46
Chapter 3: Vector Algorithms and Architectures		51
3.1	Vector and Matrix Algorithms	52
3.2	A Vector Architecture—Single Instruction Multiple Data	58
3.3	An SIMD Instruction Set	63
3.3.1	Registers and Memories of an SIMD Computer	64
3.3.2	Vector, Control Unit, and Cooperative Instructions	66
3.3.3	Data-Dependent Conditional Operations	69
3.3.4	Vector Length and Strip Mining	72
3.3.5	Routing Data Among the PEs	74
3.4	The Prime Memory System	76
3.5	Use of the PE Index to Solve Storage Layout Problems	79
3.6	SIMD Language Constructs—Fortran 90	81
3.6.1	Arrays and Array Sections	81
3.6.2	Array Assignment and Array Expressions	83
3.6.3	Fortran 90 Array Intrinsic Functions	84
3.6.4	Examples of SIMD Operations in Fortran 90	86
3.7	Pipelined SIMD Vector Computers	89
3.7.1	Pipelined SIMD Processor Structure	90
	Processor/Memory Interaction	91
	Number and Types of Pipelines	92
	Implementation of Arithmetic	93
3.7.2	The Memory Interface of a Pipelined SIMD Computer	94
3.7.3	Performance of Pipelined SIMD Computers	96
3.8	Vector Architecture Summary	99
3.9	Bibliographic Notes	100
Chapter 4: MIMD Computers or Multiprocessors		109
4.1	Shared Memory and Message-Passing Architectures	110
4.1.1	Mixed-Type Multiprocessor Architectures	111
4.1.2	Characteristics of Shared Memory and Message Passing	112

4.1.3	Switching Topologies for Message Passing Architectures	114
4.1.4	Direct and Indirect Networks	116
4.1.5	Classification of Real Systems	117
4.2	Overview of Shared Memory Multiprocessor Programming	118
4.2.1	Data Sharing and Process Management	119
4.2.2	Synchronization	121
4.2.3	Atomicity and Synchronization	121
4.2.4	Work Distribution	123
4.2.5	Many Processes Executing One Program	124
4.3	Shared Memory Programming Alternatives and Scope	126
4.3.1	Process Management—Starting, Stopping, and Hierarchy	127
4.3.2	Data Access by Parallel Processes	128
4.3.3	Work Distribution	130
4.3.4	Multiprocessor Synchronization	135
	Atomicity	135
	Hardware and Software Synchronization Mechanisms	138
	Fairness and Mutual Exclusion	140
4.4	A Shared Memory Multiprocessor Programming Language	141
4.4.1	The OpenMP Language Extension	141
	Execution Model	141
	Process Control	142
	Parallel Scope of Variables	142
	Work Distribution	143
	Synchronization and Memory Consistency	143
4.4.2	The OpenMP Fortran Applications Program Interface (API)	144
	Constructs of the OpenMP Fortran API	146
4.4.3	OpenMP Fortran Examples and Discussion	151
4.5	Pipelined MIMD—Multithreading	158
4.6	Summary and Conclusions	161
4.7	Bibliographic Notes	163
	Chapter 5: Distributed Memory Multiprocessors	171
5.1	Distributing Data and Operations Among Processor/Memory Pairs	172
5.2	Programming with Message Passing	174
5.2.1	The Communicating Sequential Processes (CSP) Language	176
5.2.2	A Distributed Memory Programming Example: Matrix Multiply	180
5.3	Characterization of Communication	183
5.3.1	Point-to-Point Communications	183

5.3.2	Variable Classes in a Distributed Memory Program	186
5.3.3	High-Level Communication Operations	188
5.3.4	Distributed Gauss Elimination with High-Level Communications	191
5.3.5	Process Topology Versus Processor Topology	194
5.4	The Message Passing Interface, MPI	198
5.4.1	Basic Concepts in MPI	199
	Communicator Structure	199
	The Envelope	200
	The Data	201
	Point-to-Point Communication Concepts	201
	Collective Communications Concepts	202
5.4.2	An Example MPI Program—Matrix Multiplication	203
5.5	Hardware Managed Communication—Distributed Cache	210
5.5.1	Cache Coherence	210
5.5.2	Shared Memory Consistency	212
5.6	Conclusion—Shared Versus Distributed Memory Multiprocessors	215
5.7	Bibliographic Notes	218
	Chapter 6: Interconnection Networks	223
6.1	Network Characteristics	224
6.2	Permutations	228
6.3	Static Networks	232
6.3.1	Mesh	232
6.3.2	Ring	235
6.3.3	Tree	236
6.3.4	Cube Networks	238
6.3.5	Performance	246
6.4	Dynamic Networks	246
6.4.1	Bus	247
6.4.2	Crossbar	247
6.4.3	Multistage Interconnection Networks (MINs)	248
	Benes Network	248
	Butterfly Network	250
	Omega Network	252
6.4.4	Combining Networks—Mutual Exclusion Free Synchronization	255
6.4.5	Performance	260
6.5	Conclusion	264
6.6	Bibliographic Notes	265

Chapter 7: Data Dependence and Parallelism	269
7.1 Discovering Parallel Operations in (Sequential) Code	270
7.2 Variables with Complex Names	273
7.2.1 Nested Loops	275
7.2.2 Variations on the Array Reference Disambiguation Problem	277
7.3 Sample Compiler Techniques	282
7.3.1 Loop Transformations	282
7.3.2 Loop Restructuring	285
7.3.3 Loop Replacement Transformations	287
7.3.4 Anti- and Output Dependence Removal Transformations	290
7.4 Data Flow Principles	292
7.4.1 Data Flow Concepts	293
7.4.2 Graphical Representation of Data Flow Computations	295
7.4.3 Data Flow Conditionals	297
7.4.4 Data Flow Iteration	299
7.4.5 Data Flow Function Application and Recursion	301
7.4.6 Structured Values in Data Flow—Arrays	304
7.5 Data Flow Architectures	310
7.5.1 The MIT Static Data Flow Architecture	311
7.5.2 Dynamic Data Flow Computers	314
Manchester Data Flow Computer	315
The MIT Tagged-Token Data Flow Machine	315
7.5.3 Issues to Be Addressed by Data Flow Machines	318
7.6 Systolic Arrays	319
7.7 Conclusion	326
7.8 Bibliographic Notes	326
Chapter 8: Implementing Synchronization and Data Sharing	331
8.1 The Character of Information Conveyed by Synchronization	332
8.2 Synchronizing Different Kinds of Cooperative Computations	333
8.2.1 One Producer with One or More Consumers	334
8.2.2 Global Reduction	334
8.2.3 Global Prefix	336
8.2.4 Cooperative Update of a Partitioned Structure	338
8.2.5 Managing a Shared Task Set	338
8.2.6 Cooperative List Manipulation	339
8.2.7 Parallel Access Queue Using Fetch&add	341