

经 典 原 版 书 库

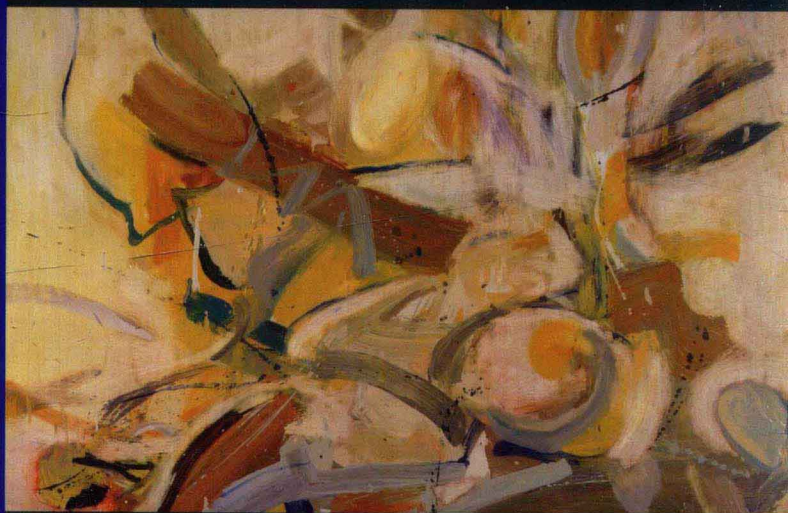
# 多处理器编程的艺术

(美) Maurice Herlihy Nir Shavit 著  
布朗大学 麻省理工学院

(英文版·修订版)

REVISED FIRST EDITION

## THE ART of MULTIPROCESSOR PROGRAMMING



机械工业出版社  
China Machine Press

MK  
MORGAN KAUFMANN

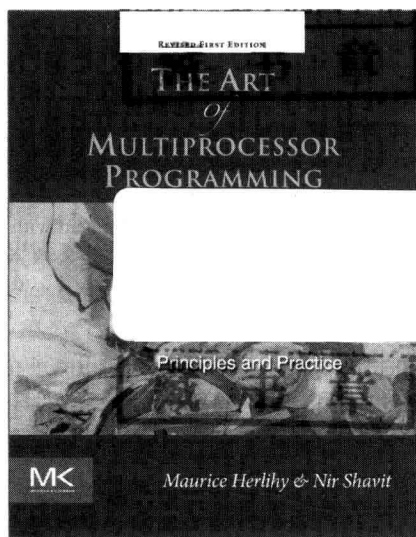
Maurice Herlihy & Nir Shavit

经 典 原 版 书 库

# 多处理器编程的艺术

(英文版·修订版)

*The Art of Multiprocessor Programming* (Revised First Edition)



(美) Maurice Herlihy Nir Shavit 著  
布朗大学 麻省理工学院



机械工业出版社  
China Machine Press

## 图书在版编目 ( CIP ) 数据

多处理器编程的艺术 ( 英文版·修订版) / ( 美) 赫利希 ( Herlihy, M. ), ( 美) 谢菲特 ( Shavit, N. ) 著.  
—北京: 机械工业出版社, 2013.1

( 经典原版书库 )

书名原文: The Art of Multiprocessor Programming, Revised First Edition

ISBN 978-7-111-41233-5

I. 多… II. ① 赫… ② 谢… III. 微处理器—程序设计—英文 IV. TP332

中国版本图书馆 CIP 数据核字 ( 2013 ) 第 012568 号

### 版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问 北京市展达律师事务所

本书版权登记号: 图字: 01-2012-7894

The Art of Multiprocessor Programming, Revised First Edition

Maurice Herlihy and Nir Shavit

ISBN: 978-0-12-397337-5

Copyright © 2012 by Elsevier Inc. All rights reserved.

Authorized English language reprint edition published by the Proprietor.

Copyright © 2013 by Elsevier (Singapore) Pte Ltd. All rights reserved.

Elsevier (Singapore) Pte Ltd.

3 Killiney Road

#08-01 Winsland House I

Singapore 239519

Tel: (65) 6349-0200

Fax: (65) 6733-1817

First Published 2013

Printed in China by China Machine Press under special arrangement with Elsevier (Singapore) Pte Ltd.

This edition is authorized for sale in China only, excluding Hong Kong SAR, Macau 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Elsevier (Singapore) Pte Ltd. 授权机械工业出版社在中国大陆境内独家发行。本版仅限在中国境内 ( 不包括香港特别行政区、澳门特别行政区及台湾地区 ) 出版及标价销售。未经许可之出口, 视为违反著作权法, 将受法律之制裁。

本书封底贴有Elsevier防伪标签, 无标签者不得销售。

机械工业出版社 ( 北京市西城区百万庄大街22号 邮政编码 100037 )

责任编辑: 迟振春

北京瑞德印刷有限公司印刷

2013年2月第1版第1次印刷

170mm×242mm·33印张

标准书号: ISBN 978-7-111-41233-5

定价: 79.00元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: ( 010 ) 88378991 88361066

投稿热线: ( 010 ) 88379604

购书热线: ( 010 ) 68326294 88379649 68995259 读者信箱: hzsj@hzbook.com

# 出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与 Pearson, McGraw-Hill, Elsevier, MIT, John Wiley & Sons, Cengage 等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出 Andrew S. Tanenbaum, Bjarne Stroustrup, Brain W. Kernighan, Dennis Ritchie, Jim Gray, Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, Abraham Silberschatz, William Stallings, Donald E. Knuth, John L. Hennessy, Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方式如下：

华章网站：[www.hzbook.com](http://www.hzbook.com)

电子邮件：[hzjsj@hzbook.com](mailto:hzjsj@hzbook.com)

联系电话：(010) 88379604

联系地址：北京市西城区百万庄南街1号

邮政编码：100037



华章科技图书出版中心

This book offers complete code for all the examples, as well as slides, updates, and other useful tools on its companion web page at: <http://store.elsevier.com/product.jsp?isbn=9780123973375>

# Preface

This book is intended to serve both as a textbook for a senior-level undergraduate course, and as a reference for practitioners.

Readers should know enough discrete mathematics to understand “big-O” notation, and what it means for a problem to be NP-complete. It is helpful to be familiar with elementary systems constructs such as processors, threads, and caches. A basic understanding of Java is needed to follow the examples. (We explain advanced language features before using them.) Two appendixes summarize what the reader needs to know: Appendix A covers programming language constructs, and Appendix B covers multiprocessor hardware architectures.

The first third covers the *principles* of concurrent programming, showing how to *think* like a concurrent programmer. Like many other skills such as driving a car, cooking a meal, or appreciating caviar, thinking concurrently requires cultivation, but it can be learned with moderate effort. Readers who want to start programming right away may skip most of this section, but should still read Chapters 2 and 3 which cover the basic ideas necessary to understand the rest of the book.

We first look at the classic *mutual exclusion* problem (Chapter 2). This chapter is essential for understanding why concurrent programming is a challenge. It covers basic concepts such as fairness and deadlock. We then ask what it means for a concurrent program to be correct (Chapter 3). We consider several alternative conditions, and the circumstances one might want to use each one. We examine the properties of *shared memory* essential to concurrent computation (Chapter 4), and we look at the kinds of synchronization primitives needed to implement highly concurrent data structures (Chapters 5 and 6).

We think it is essential that anyone who wants to become truly skilled in the art of multiprocessor programming spend time solving the problems presented in the first part of this book. Although these problems are idealized, they distill



the kind of thinking necessary to write effective multiprocessor programs. Most important, they distill the style of thinking necessary to avoid the common mistakes committed by nearly all novice programmers when they first encounter concurrency.

The next two-thirds describe the *practice* of concurrent programming. Each chapter has a secondary theme, illustrating either a particular programming pattern or algorithmic technique. At the level of systems and languages, Chapter 7 covers spin locks and contention. This chapter introduces the importance of the underlying architecture, since spin lock performance cannot be understood without understanding the multiprocessor memory hierarchy. Chapter 8 covers monitor locks and waiting, a common synchronization idiom, especially in Java. Chapter 16 covers work-stealing and parallelism, and Chapter 17 describes barriers, all of which are useful for structuring concurrent applications.

Other chapters cover concurrent data structures. All these chapters depend on Chapter 9, and the reader should read this chapter before reading the others. Linked lists illustrate different kinds of synchronization patterns, ranging from coarse-grained locking, to fine-grained locking, to lock-free structures (Chapter 9). The FIFO queues illustrate the ABA synchronization hazard that arises when using atomic synchronization primitives (Chapter 10), Stacks illustrate an important synchronization pattern called *elimination* (Chapter 11), Hash maps show how an algorithm can exploit natural parallelism (Chapter 13), Skip lists illustrate efficient parallel search (Chapter 14), and priority queues illustrate how one can sometimes weaken correctness guarantees to enhance performance (Chapter 15).

Finally, Chapter 18 describes the emerging *transactional* approach to concurrency, which we believe will become increasingly important in the near future.

The importance of concurrency has not always been acknowledged. Here is a quote from a 1989 *New York Times* article on new operating systems for the IBM PC:

Real concurrency—in which one program actually continues to function while you call up and use another—is more amazing but of small use to the average person. How many programs do you have that take more than a few seconds to perform any task?

Read this book, and decide for yourself.

# Acknowledgments

We would like to thank Doug Lea, Michael Scott, Ron Rivest, Tom Corman, Radia Perlman, George Varghese and Michael Sipser for their help in finding the right publication venue for our book.

We thank all the students, colleagues, and friends who read our draft chapters and sent us endless lists of comments and ideas: Yehuda Afek, Shai Ber, Martin Buchholz, Vladimir Budovsky, Christian Cachin, Cliff Click, Yoav Cohen, Dave Dice, Alexandra Fedorova, Pascal Felber, Christof Fetzer, Shafi Goldwasser, Rachid Guerraoui, Tim Harris, Danny Hendler, Maor Hizkiev, Eric Koskinen, Christos Kozyrakis, Edya Ladan, Doug Lea, Oren Lederman, Pierre Leone, Yossi Lev, Wei Lu, Victor Luchangco, Virendra Marathe, Kevin Marth, John Mellor-Crummey, Mark Moir, Dan Nussbaum, Kiran Pamnany, Ben Pere, Torvald Riegel, Vijay Saraswat, Bill Scherer, Warren Schudy, Michael Scott, Ori Shalev, Marc Shapiro, Yotam Soen, Ralf Suckow, Seth Syberg, Alex Weiss, and Zhenyuan Zhao. We apologize for any names inadvertently omitted.

We thank Mark Moir, Steve Heller, and our colleagues in the Scalable Synchronization group at Sun Microsystems for their incredible support during the writing of the book.

Thanks to all who have sent us errata to improve this book, including: Rajeev Alur, Matthew Allen, Karolos Antoniadis, Cristina Basescu, Liran Bارسا, Igor Berman, Konstantin Boudnik, Bjoern Brandenburg, Martin Buchholz, Kyle Cackett, Mario Calha, Michael Champigny, Neill Clift, Eran Cohen, Daniel B. Curtis, Gil Danziger, Venkat Dhinakaran, David Dice, Wan Fokkink, David Fort, Robert P. Goddard, Brian Goetz, Bart Golsteijn, K. Gopinath, Enes Goktas, Jason T. Greene, Dan Grossman, Tim Halloran, Muhammad Amber Hassaan, Matt Hayes, Francis Hools, Ben Horowitz, Barak Itkin, Paulo Janotti, Kyungho Jeon, Ahmed Khademzadeh, Irena Karlinsky, Habib Khan, Omar



Khan, Namhyung Kim, Guy Korland, Sergey Kotov, Doug Lea, Yossi Lev, Adam MacBeth, Kevin Marth, Adam Morrison, Adam Weinstock, Mike Maloney, Tim McIver, Sergejs Melderis, Bartosz Milewski, Mark Moir, Adam Morrison, Victor Luchangco, Jose Pedro Oliveira, Dale Parson, Jonathan Perry, Amir Pnueli, Pat Quillen, Binoy Ravindran, Roei Raviv, Sudarshan Raghunathan, Jean-Paul Rigault, Michael Rueppel, Mohamed M. Saad, Assaf Schuster, Marc Shapiro, Nathar Shah, Huang-Ti Shih, Joseph P. Skudlarek, James Stout, Mark Summerfield, Deqing Sun, Seth Syberg, Fuad Tabb, Binil Thomas, John A Trono, Thomas Weibel, Adam Weinstock, Jaeheon Yi, Zhenyuan Zhao, Ruiwen Zuo, Chong Xing.

# Suggested Ways to Teach the Art of Multiprocessor Programming

## Preface

The following are three possible tracks to teaching a multiprocessor programming course using the material in the book.

The first track is a short course for *practitioners* who are interested in techniques that can be applied directly to problems at hand.

The second track is a longer course for students who are *not Computer Science majors*, but who are interested in learning the basics of multiprocessor programming, as well as techniques likely to be useful in their own areas.

The third track is a semester-long course for *Computer Science majors*, either upper-level undergraduates or graduate students.

## Practitioner Track

Cover Chapter 1, emphasizing Amdahl's law and its implications. In Chapter 2, cover sections 2.1, 2.2, 2.3, and 2.6. Mention the *implications* of the impossibility proofs in Section 2.8. In Chapter 3, skip Sections 3.3 and 3.6.

Cover Chapter 7, except for Sections 7.7 and 7.8. Chapter 8, which deals monitors and reentrant locks, may be familiar to some practitioners. Skip Section 8.5 on Semaphores.

Cover Chapters 9, 10, except for 10.7, and Sections 11.1 and 11.2. Skip the material in Sections 11.3 and onwards. Skip Chapter 12.

Cover Chapters 13 and 14. Skip Chapter 15. Cover Chapter 16, except for Section 16.5. In Chapter 17, teach sections 17.1–17.3.

## **Non-CS Major Track**

Cover Chapter 1, emphasizing Amdahl's law and its implications. In Chapter 2, cover sections 2.1, 2.2, 2.3, 2.5, and 2.6. Mention the *implications* of the impossibility proofs in Section 2.8. In Chapter 3 skip Section 3.6.

Cover the material in Sections 4.1 and 4.2, and Chapter 5. Mention the universality of consensus, but skip Chapter 6.

Cover Chapter 7, except for Sections 7.7 and 7.8. Cover Chapter 8.

Cover Chapters 9, 10, except for 10.7, and Chapter 11. Skip Chapter 12.

Cover Chapters 13 and 14. Skip Chapter 15. Cover Chapter 16. In Chapter 17, teach sections 17.1–17.3. Cover Chapter 18.

## **CS Major Track**

The slides on the companion page were developed for a semester-long course

Cover Chapters 1 and 2 (Section 2.7 is optional) and 3. (Section 3.6 is optional). Cover Chapters 4, 5, and 6. Before starting Chapter 7, it may be useful to review basic multiprocessor architecture (Appendix B).

Cover Chapter 7 (Sections 7.7 and 7.8 are optional). Cover Chapter 8 if your students are unfamiliar with Java monitors. Cover Chapters 9 and 10 (Section 10.7 optional). Cover Chapters 11, 12 (Sections 12.7, 12.8, and 12.9 are optional), 13, and 14. Chapter 15 is optional. Cover Chapters 16 and 17. Chapter 18 is optional.


# Contents

Preface	v
Acknowledgments	vii
Suggested Ways to Teach the Art of Multiprocessor Programming	ix
<b>I Introduction</b>	<b>I</b>
1.1 Shared Objects and Synchronization	3
1.2 A Fable	6
1.2.1 Properties of Mutual Exclusion	8
1.2.2 The Moral	9
1.3 The Producer–Consumer Problem	10
1.4 The Readers–Writers Problem	12
1.5 The Harsh Realities of Parallelization	13
1.6 Parallel Programming	15
1.7 Chapter Notes	15
1.8 Exercises	16
<b>PRINCIPLES</b>	<b>19</b>
<b>2 Mutual Exclusion</b>	<b>21</b>
2.1 Time	21

2.2	Critical Sections	22
2.3	2-Thread Solutions	24
2.3.1	The LockOne Class	25
2.3.2	The LockTwo Class	26
2.3.3	The Peterson Lock	27
2.4	The Filter Lock	28
2.5	Fairness	31
2.6	Lamport's Bakery Algorithm	31
2.7	Bounded Timestamps	33
2.8	Lower Bounds on the Number of Locations	37
2.9	Chapter Notes	40
2.10	Exercises	41
<b>3</b>	<b>Concurrent Objects</b>	<b>45</b>
3.1	Concurrency and Correctness	45
3.2	Sequential Objects	48
3.3	Quiescent Consistency	49
3.3.1	Remarks	51
3.4	Sequential Consistency	51
3.4.1	Remarks	52
3.5	Linearizability	54
3.5.1	Linearization Points	55
3.5.2	Remarks	55
3.6	Formal Definitions	55
3.6.1	Linearizability	57
3.6.2	Compositional Linearizability	57
3.6.3	The Nonblocking Property	58
3.7	Progress Conditions	59
3.7.1	Dependent Progress Conditions	60
3.8	The Java Memory Model	61
3.8.1	Locks and Synchronized Blocks	62
3.8.2	Volatile Fields	63
3.8.3	Final Fields	63

3.9 Remarks	64
3.10 Chapter Notes	65
3.11 Exercises	66
<b>4 Foundations of Shared Memory</b>	<b>71</b>
4.1 The Space of Registers	72
4.2 Register Constructions	77
4.2.1 MRSW Safe Registers	78
4.2.2 A Regular Boolean MRSW Register	78
4.2.3 A Regular $M$ -Valued MRSW Register	79
4.2.4 An Atomic SRSW Register	81
4.2.5 An Atomic MRSW Register	82
4.2.6 An Atomic MRMW Register	85
4.3 Atomic Snapshots	87
4.3.1 An Obstruction-Free Snapshot	87
4.3.2 A Wait-Free Snapshot	88
4.3.3 Correctness Arguments	90
4.4 Chapter Notes	93
4.5 Exercises	94
<b>5 The Relative Power of Primitive Synchronization Operations</b>	<b>99</b>
5.1 Consensus Numbers	100
5.1.1 States and Valence	101
5.2 Atomic Registers	103
5.3 Consensus Protocols	106
5.4 FIFO Queues	106
5.5 Multiple Assignment Objects	110
5.6 Read–Modify–Write Operations	112
5.7 Common2 RMW Operations	114
5.8 The <code>compareAndSet()</code> Operation	116



5.9 Chapter Notes	117
5.10 Exercises	118
<b>6 Universality of Consensus</b>	<b>125</b>
6.1 Introduction	125
6.2 Universality	126
6.3 A Lock-Free Universal Construction	126
6.4 A Wait-Free Universal Construction	130
6.5 Chapter Notes	136
6.6 Exercises	137
 <b>PRACTICE</b>	<b>139</b>
<b>7 Spin Locks and Contention</b>	<b>141</b>
7.1 Welcome to the Real World	141
7.2 Test-And-Set Locks	144
7.3 TAS-Based Spin Locks Revisited	146
7.4 Exponential Backoff	147
7.5 Queue Locks	149
7.5.1 Array-Based Locks	150
7.5.2 The CLH Queue Lock	151
7.5.3 The MCS Queue Lock	154
7.6 A Queue Lock with Timeouts	157
7.7 A Composite Lock	159
7.7.1 A Fast-Path Composite Lock	165
7.8 Hierarchical Locks	167
7.8.1 A Hierarchical Backoff Lock	167
7.8.2 A Hierarchical CLH Queue Lock	168
7.9 One Lock To Rule Them All	173
7.10 Chapter Notes	173
7.11 Exercises	174

<b>8 Monitors and Blocking Synchronization</b>	<b>177</b>
8.1 Introduction	177
8.2 Monitor Locks and Conditions	178
8.2.1 Conditions	179
8.2.2 The Lost-Wakeup Problem	181
8.3 Readers–Writers Locks	183
8.3.1 Simple Readers–Writers Lock	184
8.3.2 Fair Readers–Writers Lock	185
8.4 Our Own Reentrant Lock	187
8.5 Semaphores	189
8.6 Chapter Notes	189
8.7 Exercises	190
<b>9 Linked Lists: The Role of Locking</b>	<b>195</b>
9.1 Introduction	195
9.2 List-Based Sets	196
9.3 Concurrent Reasoning	198
9.4 Coarse-Grained Synchronization	200
9.5 Fine-Grained Synchronization	201
9.6 Optimistic Synchronization	205
9.7 Lazy Synchronization	208
9.8 Non-Blocking Synchronization	213
9.9 Discussion	218
9.10 Chapter Notes	219
9.11 Exercises	219
<b>10 Concurrent Queues and the ABA Problem</b>	<b>223</b>
10.1 Introduction	223
10.2 Queues	224
10.3 A Bounded Partial Queue	225
10.4 An Unbounded Total Queue	229
10.5 An Unbounded Lock-Free Queue	230

10.6	Memory Reclamation and the ABA Problem	233
10.6.1	A Naïve Synchronous Queue	237
10.7	Dual Data Structures	238
10.8	Chapter Notes	241
10.9	Exercises	241
<b>11</b>	<b>Concurrent Stacks and Elimination</b>	<b>245</b>
11.1	Introduction	245
11.2	An Unbounded Lock-Free Stack	245
11.3	Elimination	248
11.4	The Elimination Backoff Stack	249
11.4.1	A Lock-Free Exchanger	249
11.4.2	The Elimination Array	251
11.5	Chapter Notes	254
11.6	Exercises	255
<b>12</b>	<b>Counting, Sorting, and Distributed Coordination</b>	<b>259</b>
12.1	Introduction	259
12.2	Shared Counting	259
12.3	Software Combining	260
12.3.1	Overview	261
12.3.2	An Extended Example	267
12.3.3	Performance and Robustness	269
12.4	Quiescently Consistent Pools and Counters	269
12.5	Counting Networks	270
12.5.1	Networks That Count	270
12.5.2	The Bitonic Counting Network	273
12.5.3	Performance and Pipelining	280
12.6	Diffraction Trees	282
12.7	Parallel Sorting	286
12.8	Sorting Networks	286
12.8.1	Designing a Sorting Network	287
12.9	Sample Sorting	290
12.10	Distributed Coordination	291