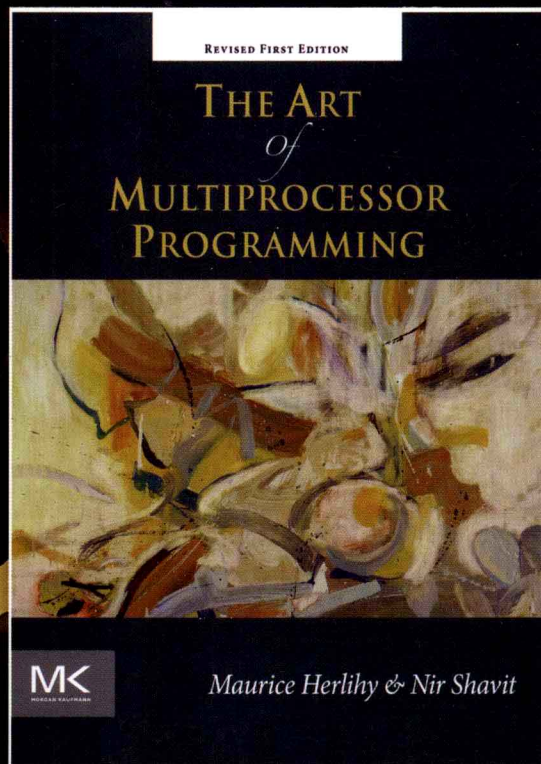


多处理器编程的艺术 (修订版)

(美) Maurice Herlihy Nir Shavit 著 金海 胡侃 译
布朗大学 麻省理工学院 华中科技大学

The Art of Multiprocessor Programming
Revised First Edition

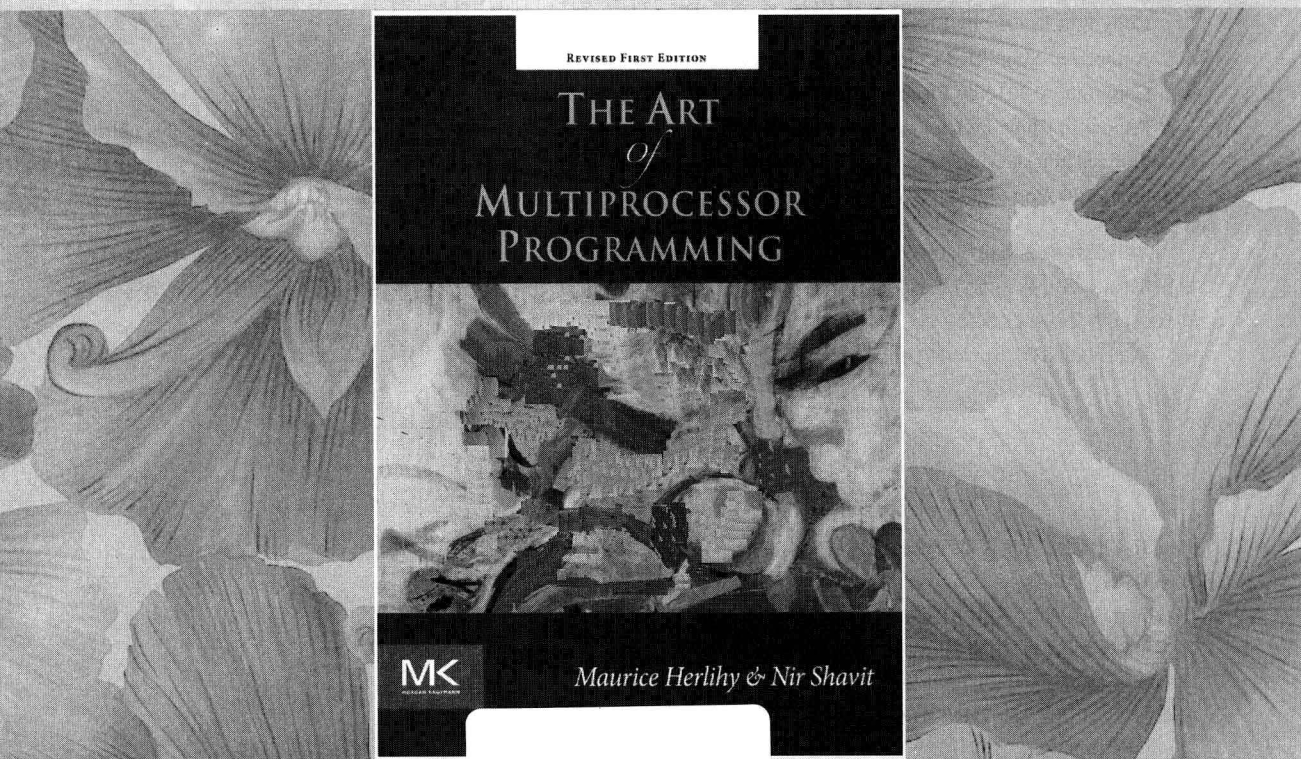


计 算 机 科 学 丛

多处理器编程的艺术 (修订版)

(美) Maurice Herlihy Nir Shavit 著 金海 胡侃 译
布朗大学 麻省理工学院 华中科技大学

The Art of Multiprocessor Programming
Revised First Edition



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 (修订版) / (美) 赫利希 (Herlihy, M.), (美) 谢菲特 (Shavit, N.) 著; 金海, 胡侃译. —北京: 机械工业出版社, 2013.4

(计算机科学丛书)

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

ISBN 978-7-111-41858-0

I. 多… II. ① 赫… ② 谢… ③ 金… ④ 胡… III. 微处理器—程序设计 IV. TP332

中国版本图书馆CIP数据核字 (2013) 第055426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: 01-2012-7897

本书从原理和实践两个方面全面阐述了多处理器编程的指导原则, 包含编制高效的多处理器程序所必备的算法技术。此外, 附录提供了采用其他程序设计语言包 (如C#、C及C++的PThreads库) 进行编程的相关背景知识以及硬件基础知识。

本书适合作为高等院校计算机及相关专业高年级本科生及研究生的教材, 同时也可作为相关技术人员的参考书。

Maurice Herlihy and Nir Shavit: The Art of Multiprocessor Programming, Revised First Edition (ISBN 978-0-12-397337-5).

Copyright © 2012 by Elsevier Inc. All rights reserved.

Authorized Simplified Chinese translation edition published by the Proprietor.

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

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 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.

本书简体中文版由Elsevier(Singapore)Pte Ltd. 授权机械工业出版社在中国大陆境内独家出版和发行。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

藁城市京瑞印刷有限公司印刷

2013年5月第1版第1次印刷

185mm×260mm · 24.75印张

标准书号: ISBN 978-7-111-41858-0

定 价: 69.00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: hzjsj@hzbook.com

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，不仅肇划了研究的范畴，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, Brian 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

电子邮件：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



每逢我们在多处理器平台上进行编程时，往往会有这么一种感觉：即使已熟练掌握了系统提供的各种同步原语，但所编制的并行程序的实际性能似乎总有些差强人意，并不十分理想。究其原因，问题的根结在于多处理器编程应是一门科学和艺术完美结合的学科。若要在多处理器系统结构上编制出性能良好的并行程序，要求设计者不仅要精通多处理器系统结构、并行算法以及一些系统构建工具，还应能基于一种设计理念，充分发挥个人的想象空间，合理搭配这些知识和资源，从而和谐地构建完整的系统，使设计者能比底层硬件和操作系统“做得更好”。也就是说，在编写多处理器程序时，要能同时从宏观和微观两种角度分析问题，并能在这两种角度之间灵活地转换。

自20世纪中叶第一台通用电子计算机研制成功以来，程序的编制大多是基于顺序计算模型的，程序的执行过程是操作的有序序列。由于顺序计算机能够用图灵机精确地描述，因此顺序计算的编程能在一组易于理解且完备定义的抽象之上进行，而不需要了解底层的细节。近年来，尽管单处理器仍在发展，但由于指令级并行的开发空间正在减少，再加上散热等问题限制了时钟频率的继续提高，所以单处理器发展的速度正在减缓，这最终导致了起源于在单独一个晶片上设计多个内核的多处理器系统结构的出现。多处理器系统结构允许多个处理器执行同一个程序，共享同一程序的代码和地址空间，并利用并行技术来提高计算效率。在这种计算模型中，并发程序的执行可以看做是多个并发线程对一组共享对象的操作序列，为了在这种异步并发环境中获得更好的性能，底层系统结构的细节需要呈现给设计者。

作为一名优秀的程序设计员，在编写多处理器程序之前首先应弄清楚：多处理器计算机的能力和限制是什么；在异步并发计算模型中什么是可解决的，什么是不可解决的；是什么使得某些问题很难计算，而又使另一些问题容易计算。这要求设计者具备一定的多核并行计算理论基础知识，掌握多处理器系统结构上并发计算模型的可计算性理论及复杂性理论。其次，应掌握基本的多核平台上的并行程序设计技术，包括并行算法、同步原语以及各种多核系统结构。

Maurice Herlihy 教授和 Nir Shavit 教授在并发程序设计领域具有很深的造诣，并拥有40年以上一起从事并发程序设计教学的合作经验。他们对多处理器并行程序设计技术做出了巨大的贡献，并因此而成为2004年ACM/EATCS哥德尔奖和2012年分布式计算领域Dijkstra奖的共同获得者。这本由他们合著的专著致力于解决如何采用更好的并行算法来克服多核并发程序并行度低的问题。

Amdahl定律早已明确地告诉我们，从程序本身可获得的并行度是有限的，加速比的提高主要取决于程序中必须增加的串行执行部分，而这部分又往往包含着具有相对较高开销的通信和协作。因此，在多处理器系统结构上，如何提高程序中必须串行部分的并行度，以及降低并行处理器中远程访问的时延是我们目前面临的两大技术挑战。这些问题的有效解决，必须依靠软件技术和硬件技术的改进和发展。本书则侧重于对前一个挑战的研究。

作者先从宏观的抽象角度出发，在一个理想化的共享存储器系统结构中研究各种并行算

法的可计算性及正确性。通过对这些经典算法的推理分析，向读者揭示了现代协作范例和并发数据结构中所隐藏的核心思想，使读者学会如何分析饥饿和死锁等微妙的活性问题，深层次地研究现代硬件同步原语所应具有的能力及其特性。随后，从微观的实际角度出发，针对当今主流的多处理器系统结构，设计了一系列完美高效的并行算法及并发数据结构，并对各种算法的效率及其机理进行了分析。所有的设计全部采用Java程序设计语言详细地描述，可以非常容易地将它们扩展到实际应用中。

本书的前6章讲述了多处理器程序设计的原理部分，着重于异步并发环境中的可计算性问题，借助于一个理想化的计算模型来阐述如何描述和证明并行程序的实际执行行为。由于其自身的特点，多处理器程序的正确性要比顺序执行程序的正确性复杂得多，书中为我们展现了一系列不同的辅助论证工具，令人有耳目一新之感。随后的11章阐述了多处理器程序设计的实践部分。由于在多处理器环境中编写程序时，底层系统结构的细节并不像编写顺序程序那样被完全隐藏在一种编程抽象中，因此，本书附录B介绍了多处理器硬件的基础知识。最后的第18章介绍了当今并发问题研究中最先进的事务方法，可以预言这种方法在今后的研究中将会越来越重要。

感谢王振飞博士在本书第13~18章翻译中所做的工作，感谢胡丽婷、袁旻昊、耿玮、蔡硕、张亮同学在本书翻译的初始资料整理中所给予的帮助。

本书可以作为高年级本科生的教材，也可以作为相关技术人员的参考书。

读者应具备一定的离散数学基础知识，能够理解“大O”符号的含义，以及它在NP完全问题中所起的作用；熟悉计算机系统的基本组成部件，如处理器、线程、高速缓存等；为了能够理解书中的实例，还需要具备初步的Java知识。（在使用这些高级程序设计语言之前，本书阐述了语言的相关特征。）书中提供两个附录以供读者参考：附录A包含程序设计语言的相关知识，附录B给出了多处理器硬件系统结构的相关内容。

本书前三分之一涵盖并发程序设计的基本原理，阐述并发程序设计的编程思想。就像掌握汽车驾驶技术、烹饪食物和品尝鱼子酱一样，并发思维也需要培养，需要适当的努力才能学好。希望立刻动手编程的读者可以跳过这部分的大多数内容，但仍需阅读第2章及第3章的内容，这两章包含了理解本书其他部分所必不可少的基本知识。

在原理部分中，首先讨论了经典的互斥问题（第2章），包括诸如公平性和死锁这样的基本概念，这对于理解并发程序设计的难点尤为重要。然后，结合并发执行和并发设计中可能出现的各种情形和开发环境，给出了并发程序正确性的定义（第3章）。接着，研究了对并发计算至关重要的共享存储器的性质（第4章）。最后，介绍了几种为实现高并发性数据结构而使用的同步原语（第5、6章）。

对于每一位渴望真正掌握多处理器编程艺术的程序设计人员来说，花上一定的时间去解决本书第一部分所提及的问题是很有必要的。虽然这些问题都是理想化的，但它们为编写高效的多处理器程序提供了非常有益的编程思想。尤为重要的是，通过问题解决中获取的思维方式，能够避免出现那些初次编写并发程序的设计人员普遍易犯的错误。

接下来的三分之二讲述并发程序设计的具体实践。每章都有一个相应主题，阐明一种特定的程序设计模式或者算法技巧。第7章从系统和语言这两个不同的抽象层面，讨论争用及自旋锁的概念，强调了底层系统结构的重要性，指出对于自旋锁性能的理解必须建立在对多处理器层次存储结构充分理解的基础上。第8章涉及等待及管程锁的概念，这是一种常用的同步用语（特别是在Java中）。第16章包括并行性及工作窃取问题，第17章则介绍了障碍技术，这种技术往往在具有并发结构的应用中得以广泛的使用。

其他章节讲述各种类型的并发数据结构。它们均以第9章的概念为基础，因此建议读者在阅读其他章节之前首先阅读第9章的内容。该章采用链表结构来阐明各种类型的同步模式，包括粗粒度锁、细粒度锁及无锁结构。第10章则借助于FIFO队列说明在使用同步原语时可能出现的ABA问题，第11章通过栈描述了一种重要的同步模式——消除，第13章通过哈希映射阐述如何利用固有并行进行算法设计。高效率的并行查找技术则借助于跳表来阐述（第14章），而如何通过降低正确性标准来获得更高的性能则通过优先级队列进行了阐述（第15章）。

最后，第18章介绍了在并发问题的研究中新出现的事务方法，可以确信这种方法在不远的将来会变得越来越重要。

并发性的重要性还没有得到人们的广泛认可。在此，引用《纽约时报》1989年关于IBM PC中新型操作系统的一段评论：

真正的并发（当你唤醒并使用另一个程序时原来的程序仍继续运行）是非常令人振奋的，但对于普通使用者来说用处却很小。您能有几个程序在执行时需要花费数秒甚至更多的时间呢？

致谢

感谢Doug Lea、Michael Scott、Ron Rivest、Tom Corman、Radia Perlman、George Varghese 和Michael Sipser在本书出版过程中所做的努力和给予的帮助。

感谢在本书的起草和修订过程中，向我们提供了大量宝贵意见的所有学生、同事和朋友：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 Hizkiv, Eric Koskinen, Christos Kozyrakis, Edya Ladan, Doug Lea, Oren Lederman, Pierre Leone, Yossi Lev, Wei Lu, Victor Luchangco, Virendra Marathe, 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和Zhenyuan Zhao。同时，也向在这里没有提及的许多朋友表示道歉和感谢。

感谢Mark Moir、Steve Heller和Sun公司的Scalable Synchronization小组成员，他们为本书的撰写提供了巨大的支持和帮助。

感谢提出勘误帮助我们改进本书的所有朋友：Rajeev Alur, Matthew Allen, Karolos Antoniadis, Cristina Basescu, Liran Barsisa, 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 Melders, 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。

教学建议

The Art of Multiprocessor Programming (Revised First Edition)

对于技术人员

技术人员对直接解决手边问题的技术感兴趣，针对技术人员的短期课程应该包括以下内容：

强调Amdahl定律及其作用的第1章。第2章的2.1、2.2、2.3和2.6节，了解关于不可能性证明的作用的2.8节。第3章的3.1、3.2、3.4、3.5、3.7、3.8和3.9节。

第7章的7.1~7.6和7.9节。对于一些技术人员来说，可能熟悉讨论管程锁和可重入锁的第8章，跳过有关信号量的8.5节。

第9章，第10章（除10.7节外），第11章的11.1和11.2节。

第13和14章，第16章（除16.5节外），第17章的17.1~17.3节。

对于非计算机专业学生

非计算机专业学生对多处理器编程的基础知识以及可能对他们自己领域有用的技术感兴趣，针对非计算机专业学生的中期课程应该包括以下内容：

强调Amdahl定律及其作用的第1章。第2章的2.1、2.2、2.3、2.5和2.6节，了解关于不可能性证明的作用的2.8节。第3章的3.1~3.5、3.7~3.9节。

第4章的4.1和4.2节，第5章。了解关于一致性的通用性的第6章。

第7章的7.1~7.6和7.9节，第8章。

第9章，第10章（除10.7节外），第11章。

第13和14章，第16章，第17章的17.1~17.3节，第18章。

对于计算机专业学生

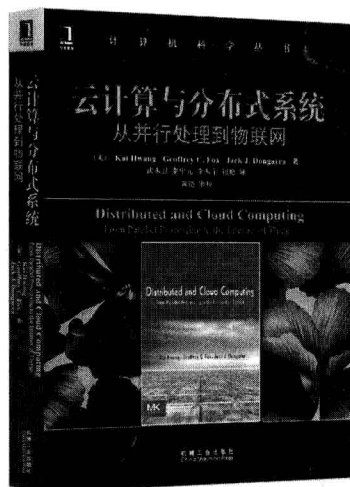
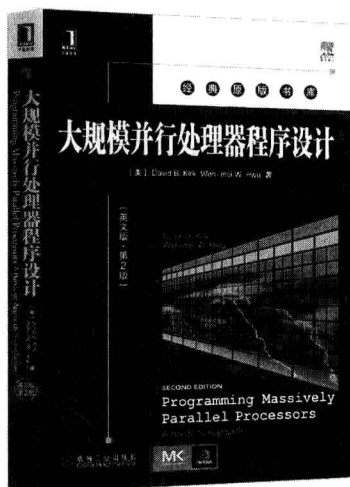
针对计算机专业学生（高年级本科生或研究生）的一学期课程应该包括以下内容：

第1章，第2章（2.7节可选），第3章（3.6节可选），第4章，第5章，第6章。在学习第7章之前，有必要复习一下多处理器系统结构的基础知识（附录B）。

第7章（7.7和7.8节可选），第8章（如果学生不熟悉Java管程知识），第9章，第10章（10.7节可选），第11章，第12章（12.7~12.9节可选），第13章，第14章。第15章可选。第16章，第17章。第18章可选。

本书的配套网站（<http://store.elsevier.com/product.jsp?isbn=9780123973375>）为一学期课程提供了授课PPT。

推荐阅读



大规模并行处理器程序设计（英文版·第2版）

作者：David B. Kirk Wen-mei W. Hwu

ISBN: 978-7-111-41629-6 定价：79.00元

云计算与分布式系统：从并行处理到物联网

作者：Kai Hwang Geoffrey C. Fox Jack Dongarra 译者：武永卫等 审校：黄锐

ISBN: 978-7-111-41065-2 定价：85.00元

并行程序设计导论

作者：Peter S. Pacheco 译者：邓倩妮等 ISBN: 978-7-111-39284-2 定价：49.00元

并行程序设计原理

作者：Larry Snyder Calvin Lin 译者：陆鑫达 ISBN: 978-7-111-27075-1 定价：45.00元

高性能CUDA应用设计与开发：方法与最佳实践

作者：Rob Farber 译者：于玉龙 唐堃 ISBN: 978-7-111-40446-0 定价：59.00元

GPU高性能编程CUDA实战

作者：Jason Sanders Edward Kandrot 译者：聂雪军等 ISBN: 978-7-111-32679-3 定价：39.00元

出版者的话	
译者序	
前言	
教学建议	
第1章 引言	1
1.1 共享对象和同步	2
1.2 生活实例	4
1.2.1 互斥特性	6
1.2.2 道德	7
1.3 生产者-消费者问题	7
1.4 读者-写者问题	9
1.5 并行的困境	9
1.6 并行程序设计	11
1.7 本章注释	11
1.8 习题	11
第一部分 原 理	
第2章 互斥	14
2.1 时间	14
2.2 临界区	14
2.3 双线程解决方案	16
2.3.1 LockOne类	16
2.3.2 LockTwo类	17
2.3.3 Peterson锁	18
2.4 过滤锁	19
2.5 公平性	21
2.6 Bakery算法	21
2.7 有界时间戳	23
2.8 存储单元数量的下界	25
2.9 本章注释	27
2.10 习题	28
第3章 并发对象	31
3.1 并发性与正确性	31
3.2 顺序对象	33
3.3 静态一致性	34
3.4 顺序一致性	35
3.5 可线性化性	38
3.5.1 可线性化点	38
3.5.2 评析	38
3.6 形式化定义	38
3.6.1 可线性化性	39
3.6.2 可线性化性的复合性	40
3.6.3 非阻塞特性	40
3.7 演进条件	41
3.8 Java存储器模型	43
3.8.1 锁和同步块	44
3.8.2 volatile域	44
3.8.3 final域	44
3.9 评析	45
3.10 本章注释	46
3.11 习题	46
第4章 共享存储器基础	50
4.1 寄存器空间	50
4.2 寄存器构造	54
4.2.1 MRSW安全寄存器	55
4.2.2 MRSW规则布尔寄存器	55
4.2.3 M-值MRSW规则寄存器	56
4.2.4 SRSW原子寄存器	57
4.2.5 MRSW原子寄存器	59
4.2.6 MRMW原子寄存器	60
4.3 原子快照	62
4.3.1 无障碍快照	63
4.3.2 无等待快照	64
4.3.3 正确性证明	66
4.4 本章注释	67
4.5 习题	67
第5章 同步原子操作的相对能力	70
5.1 一致数	70
5.2 原子寄存器	72

5.3 一致性协议	74
5.4 FIFO队列	74
5.5 多重赋值对象	77
5.6 读-改-写操作	79
5.7 Common2 RMW操作	80
5.8 compareAndSet()操作	81
5.9 本章注释	82
5.10 习题	83
第6章 一致性的通用性	87
6.1 引言	87
6.2 通用性	88
6.3 一种通用的无锁构造	88
6.4 一种通用的无等待构造	91
6.5 本章注释	95
6.6 习题	95

第二部分 实 践

第7章 自旋锁与争用	98
7.1 实际问题	98
7.2 测试-设置锁	100
7.3 再论基于TAS的自旋锁	102
7.4 指数后退	102
7.5 队列锁	104
7.5.1 基于数组的锁	104
7.5.2 CLH队列锁	106
7.5.3 MCS队列锁	107
7.6 时限队列锁	110
7.7 复合锁	112
7.8 层次锁	118
7.8.1 层次后退锁	118
7.8.2 层次CLH队列锁	119
7.9 由一个锁管理所有的锁	123
7.10 本章注释	123
7.11 习题	124
第8章 管程和阻塞同步	126
8.1 引言	126
8.2 管程锁和条件	126
8.2.1 条件	127
8.2.2 唤醒丢失问题	130
8.3 读者-写者锁	131

8.3.1 简单的读者-写者锁	131
8.3.2 公平的读者-写者锁	132
8.4 我们的可重入锁	134
8.5 信号量	135
8.6 本章注释	136
8.7 习题	136
第9章 链表：锁的作用	139
9.1 引言	139
9.2 基于链表的集合	140
9.3 并发推理	141
9.4 粗粒度同步	142
9.5 细粒度同步	143
9.6 乐观同步	146
9.7 惰性同步	149
9.8 非阻塞同步	153
9.9 讨论	157
9.10 本章注释	157
9.11 习题	158
第10章 并行队列和ABA问题	159
10.1 引言	159
10.2 队列	160
10.3 部分有界队列	160
10.4 完全无界队列	163
10.5 无锁的无界队列	164
10.6 内存回收和ABA问题	166
10.7 双重数据结构	170
10.8 本章注释	172
10.9 习题	172
第11章 并发栈和消除	174
11.1 引言	174
11.2 无锁的无界栈	174
11.3 消除	176
11.4 后退消除栈	176
11.4.1 无锁交换机	177
11.4.2 消除数组	179
11.5 本章注释	181
11.6 习题	181
第12章 计数、排序和分布式协作	184
12.1 引言	184

12.2 共享计数	184	14.3.2 算法	238
12.3 软件组合	185	14.4 无锁并发跳表	243
12.3.1 概述	185	14.4.1 简介	243
12.3.2 一个扩展实例	190	14.4.2 算法细节	245
12.3.3 性能和健壮性	191	14.5 并发跳表	251
12.4 静态一致池和计数器	192	14.6 本章注释	251
12.5 计数网	192	14.7 习题	251
12.5.1 可计数网	193	第15章 优先级队列	253
12.5.2 双调计数网	194	15.1 引言	253
12.5.3 性能和流水线	201	15.2 基于数组的有界优先级队列	253
12.6 衍射树	201	15.3 基于树的有界优先级队列	254
12.7 并行排序	204	15.4 基于堆的无界优先级队列	256
12.8 排序网	204	15.4.1 顺序堆	256
12.9 样本排序	207	15.4.2 并发堆	258
12.10 分布式协作	208	15.5 基于跳表的无界优先级队列	262
12.11 本章注释	208	15.6 本章注释	264
12.12 习题	209	15.7 习题	265
第13章 并发哈希和固有并行	212	第16章 异步执行、调度和工作分配	266
13.1 引言	212	16.1 引言	266
13.2 封闭地址哈希集	213	16.2 并行分析	271
13.2.1 粗粒度哈希集	214	16.3 多处理器的实际调度	273
13.2.2 空间分带哈希集	215	16.4 工作分配	274
13.2.3 细粒度哈希集	217	16.4.1 工作窃取	275
13.3 无锁哈希集	219	16.4.2 屈从和多道程序设计	275
13.3.1 递归有序划分	219	16.5 工作窃取双端队列	276
13.3.2 BucketList类	222	16.5.1 有界工作窃取双端队列	276
13.3.3 LockFreeHashSet<T>类	223	16.5.2 无界工作窃取双端队列	279
13.4 开放地址哈希集	225	16.5.3 工作平衡	282
13.4.1 Cuckoo哈希	225	16.6 本章注释	283
13.4.2 并发Cuckoo哈希	226	16.7 习题	284
13.4.3 空间分带的并发Cuckoo哈希	230	第17章 障碍	287
13.4.4 细粒度的并发Cuckoo哈希集	231	17.1 引言	287
13.5 本章注释	233	17.2 障碍实现	288
13.6 习题	234	17.3 语义换向障碍	288
第14章 跳表和平衡查找	235	17.4 组合树障碍	289
14.1 引言	235	17.5 静态树障碍	291
14.2 顺序跳表	235	17.6 终止检测障碍	293
14.3 基于锁的并发跳表	236	17.7 本章注释	295
14.3.1 简介	236	17.8 习题	296

第18章 事务内存	302
18.1 引言	302
18.1.1 关于锁的问题	302
18.1.2 关于compareAndSet()的问题	303
18.1.3 关于复合性的问题	304
18.1.4 我们能做什么	305
18.2 事务和原子性	305
18.3 软事务内存	306
18.3.1 事务和事务线程	309
18.3.2 僵尸事务和一致性	310
18.3.3 原子对象	311
18.3.4 如何演进	311
18.3.5 争用管理器	312
18.3.6 原子对象的实现	314

18.3.7 无干扰原子对象	315
18.3.8 基于锁的原子对象	318
18.4 硬事务内存	323
18.4.1 缓存一致性	324
18.4.2 事务缓存一致性	324
18.4.3 改进	325
18.5 本章注释	325
18.6 习题	326

第三部分 附 录

附录A 软件基础	328
附录B 硬件基础	340
参考文献	350
索引	359

引言

计算机产业正在经历着一场重大的结构重组和巨变，在没有其他变革之前，这个过程无疑将会继续进行。主要的芯片制造商，至少是现在，都纷纷放弃尝试研制速度更快的处理器。虽然摩尔定律仍旧适用：每年集成在同样大小空间中的晶体管数越来越多，然而，由于散热问题难于解决，它们的时钟速度无法继续得到提高。取而代之的做法是，制造商开始转向“多核”系统结构的研制，由多个处理器（多核）共享硬件高速缓存直接进行通信。通过将多个处理器同时分配给单一任务以获得更高的并行性，从而提高计算的效率。

多处理器系统结构的普及对计算机软件的发展带来了深刻的影响。直至今日以前，技术的进步意味着时钟速度的提升，时钟本身的加速导致了软件执行效率的提高。今天，这种搭便车的现象已不复存在，技术的进步不再指时钟速度的提升而是指并行度的提升，并行问题已经成为现代计算机科学的主要挑战。

本书着重讲述共享存储器通信方式下的多处理器编程技术。通常称这样的系统为共享存储器的多处理器，现在称之为多核。在各种规模的多处理器系统中都存在着不同的编程挑战——对于小规模的系统来说，需要协调单个芯片内的多个处理器对同一个共享存储单元的访问；对于大规模系统来说，需要协调一台超级计算机中各个处理器之间的数据路由。其次，现代计算机系统所固有的异步特征也给多处理器编程带来了挑战：在没有任何警示的情形下，系统的活动可以被各种不同的事件中止或延迟，例如中断、抢占、cache缺失和系统故障等。这种延迟现象本身是无法预测的，时延的长短也是不确定的：cache缺失可以造成不到十条指令执行时间的时延，页故障可能造成几百万条指令执行时间的时延，而操作系统抢占则会导致多达上亿条指令执行时间的时延。

本书从原理和实践这两个互补的方面阐述多处理器的程序设计。原理部分着重于可计算性理论：理解异步并发环境中的可计算问题。借助于一个理想化的计算模型，对异步并发环境中什么是可解的这一问题的进行了深入研究。在这个模型中，多个并发线程对一组共享对象进行操作，这些并发线程的对象操作序列被称为并发程序或并发算法。该模型实质上也正是Java™、C#及C++线程库中所采用的计算模型。

令人不可思议的是，的确存在一些易于说明的共享对象，我们无法采用任何并发算法来实现。因此，在编写多处理器程序之前弄清楚什么问题不能用计算机解决是十分重要的。大多数困扰多处理器程序员的问题都源自于计算模型自身的限制，所以，对并发共享存储器模型中可计算性理论的理解是学习多处理器编程必不可少的一个环节。书中与原理相关的章节向读者展现了各种各样的可计算问题，以帮助读者尽快地了解异步可计算性理论，同时也讲述了如何通过硬件和软件机制来解决这些问题。

理解可计算性的关键在于描述和证明特定程序的实际执行行为，更准确地说，即程序正确性问题。由于其自身的特点，多处理器程序的正确性比顺序程序的正确性更为复杂，因此，需要一系列不同的辅助论证工具来证明并发程序的正确性，甚至有可能只是为了“非形式化地证明”程序正确性（实际上程序员往往这么做）。顺序程序的正确性主要关心程序的各种安

全特性。安全性说明了“不好的事件”绝不会发生。例如，即使在断电时，交通指示灯也决不给任何方向显示绿灯。同样，并发程序的正确性也需要考虑安全性，但要考虑如何确保多个并发线程在各种可能的交互情形下的安全性，这使问题的解决变得难上加难。另外，还要考虑一个同样重要的因素，并发程序的正确性包括了各种各样的活性特性，而这种特性是顺序程序执行中所不会出现的。所谓活性，是指一个特定的“好的事件”一定会发生。例如，红色指示灯最终一定会变为绿灯。本书原理部分的最终目标就是要引入一系列分析推理并发程序的衡量标准和方法，为接下来研究现实对象和程序的正确性奠定基础。

本书的第二部分阐述多处理器程序设计的具体实践，着重于并发程序性能的分析。多处理器并发算法的性能分析与顺序算法的性能分析在风格上完全不同。顺序程序设计是基于一组易于理解且完备定义的抽象来进行的。编写顺序程序时，不需要了解底层的详细细节，例如，在硬盘和内存之间如何交换页面，在层次结构的高速缓存中如何移进/移出那些较小的内存单位。这种复杂的存储器层次结构实质上对程序员是不可见的，它被隐藏在一种完全的编程抽象之中。

然而在多处理器环境下，这种编程抽象被打破了，至少从性能角度来讲应该这样做。为了获得足够好的性能，程序设计人员有时需要比底层存储器系统“做得更好”，他们编写的程序代码可能让那些不熟悉多处理器系统结构的人感到莫名其妙。或许某一天，并发系统结构会像今天的顺序系统结构一样支持完全的抽象，然而到那时，程序设计人员早已了解这种新的系统结构了。

本书的原理部分讲述了一组共享对象和编程工具，着眼于每种对象和工具自身的能力，并借助于它们引出一些高层次的问题：用自旋锁来说明争用，用链表阐述数据结构设计中锁的作用等。这些问题对程序的性能都有着重要的影响，希望读者能充分理解它们，并能将所理解的内容应用于日后具体的多处理器系统设计中。最后，通过讨论诸如事务内存这种目前最先进的技术，作为本书的结束。

下面简要说明本书的写作风格。尽管有很多合适的语言可供选择，但本书仍采用了Java程序设计语言。当然，有大量的理由可以解释为何要做出这种选择，然而这样的话题还是更适于在闲暇时讨论！附录解释了Java所支持的一些概念在其他的常用语言或库中是如何表示的，同时也介绍了关于多处理器硬件的一些基础知识。纵观全书，我们尽量避免列出关于程序和算法性能的具体数据，而是从一般情形来考虑。这样做的理由是：多处理器系统之间差异很大，在一台机器上工作良好的并发程序并不代表在另一台机器上也有同样的表现，紧密结合一般情形能够保证本书所陈述的结论具有更加长久的有效性。

在每一章的末尾都提供了相关文献的引用，读者可以根据参考文献目录找到相应内容以便进一步阅读。此外，每章都提供了一些习题，读者可以据此检查自己对知识的理解程度。

1.1 共享对象和同步

在开始新工作的第一天，老板要求在一台能够支持10个并发线程的并行机上编写出查找 $1 \sim 10^{10}$ 之间素数的程序（不要考虑为什么这样做）。机器是按照分钟租用的，程序运行时间越长，花费也就越大。如果想给老板留下一个不错的印象，应该怎样去做？

在最初的尝试中，可能会为每个线程分配一个大小相等的输入域。各个线程分别找出 10^9 个数字内的素数，如图1-1所示。这种方法可能会由于一个简单但很重要的原因而最终导致失败，那就是相同大小的输入范围并不意味着相同的工作量。素数的分布是不均匀的：在 $1 \sim 10^9$ 之间有很多素数，但分布在 $9 \times 10^9 \sim 10^{10}$ 的素数却非常少。更为糟糕的是，整个范围内不同

素数的计算时间也是不相同的：判断一个较大的数是否为素数通常要比判断较小的数所花费的时间更长。简而言之，没有理由认为这种方式能够使得整个工作是由所有的线程平均承担的，甚至也不清楚哪个线程承担的工作最多。

```

1 void primePrint {
2     int i = ThreadID.get(); // thread IDs are in {0..9}
3     int block = power(10, 9);
4     for (int j = (i * block) + 1; j <= (i + 1) * block; j++) {
5         if (isPrime(j))
6             print(j);
7     }
8 }

```

图1-1 通过等分输入域达到负载平衡。对{0..9}中的每个线程分配同样大小的输入子集

在线程间划分工作的另一种可行方案就是为每个线程一次分配一个整数（图1-2）。当一个线程结束对该整数的判断后，再次请求分配另一个整数。为此，需要引入一个共享计数器对象。该对象将一个整型值封装起来，线程通过调用`getAndIncrement()`方法对该整型值进行自增操作，并返回未被增加前的先前值。

```

1 Counter counter = new Counter(1); // shared by all threads
2 void primePrint {
3     long i = 0;
4     long limit = power(10, 10);
5     while (i < limit) { // loop until all numbers taken
6         i = counter.getAndIncrement(); // take next untaken number
7         if (isPrime(i))
8             print(i);
9     }
10 }

```

图1-2 通过共享计数器达到负载平衡。每个线程对动态确定的数字进行判断

图1-3是Counter对象的Java实现。该计数器由单线程调用时效果很好，但由多线程共享使用时却会出现错误。其问题的根本就在于语句

```
return value++;
```

实质上是下面几行代码的缩写方式：

```

long temp = value;
value = temp + 1;
return temp;

```

在这段代码中，`value`是对象Counter的一个域，它被所有的线程所共享。但是，每个线程都有一个它自己的本地拷贝`temp`，该拷贝是线程的局部变量。

假设线程A和线程B同时调用Counter的`getAndIncrement()`方法。那么，A和B有可能同时从`value`中读入1，然后分别将各自的局部变量`temp`设置为1，再将`value`设置为2，最终，A和B都返回了`value`的先前值1。显然，这种情形并不是我们所期望的：对计数器`getAndIncrement()`方法的并发调用返回了同一个值。我们希望不同的线程返回不同的值。事实上，还有可能出现更坏的情形：一个线程从`value`中读入了1，在它把`value`置为2之前，另一个线程执行了多次增量循环，读入1设置为2，接着读入2设置为3。当第一个线程最终完