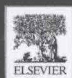


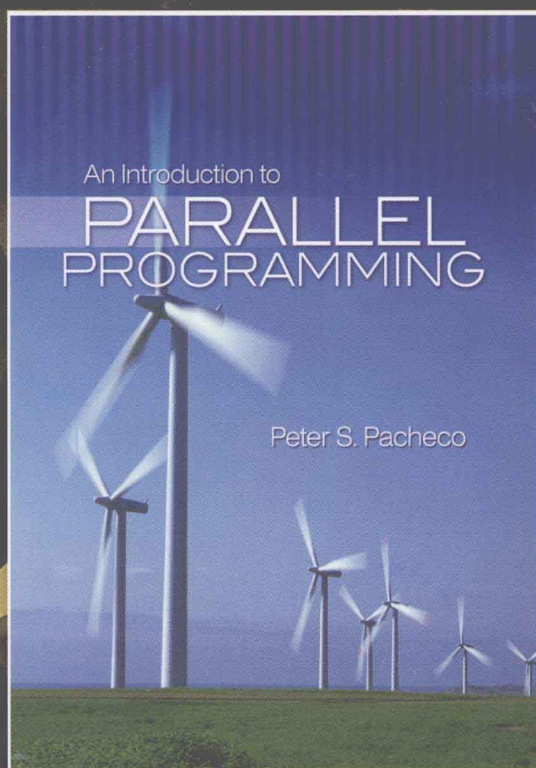
ELSEVIER

计算机科学丛书

并行程序设计导论

（美）Peter S. Pacheco 著　邓倩妮 等译

An Introduction to Parallel Programming



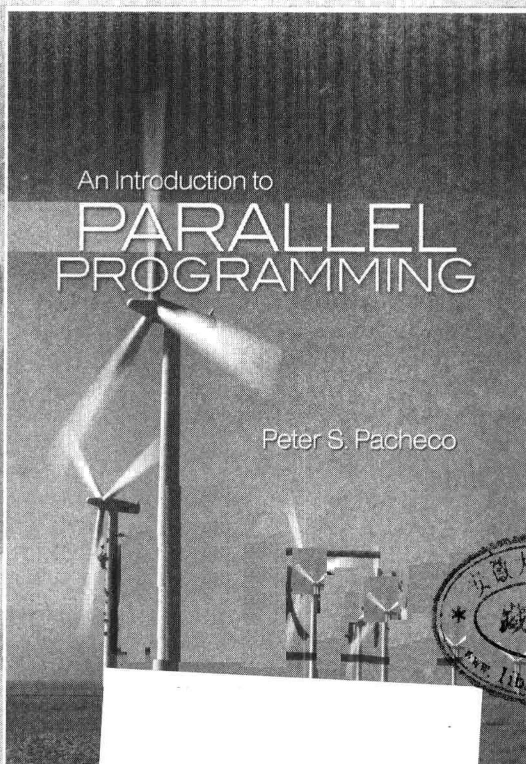
机械工业出版社
China Machine Press

机 科 学 丛 书

并行程序设计导论

（美）Peter S. Pacheco 著　　邓倩妮 等译

An Introduction to Parallel Programming



安徽大学图书馆
藏 书
www.lib.ahu.edu.cn

机械工业出版社
China Machine Press

本书全面涵盖了并行软件和硬件的方方面面，深入浅出地介绍如何使用 MPI（分布式内存编程）、Pthreads 和 OpenMP（共享内存编程）编写高效的并行程序。各章节包含了难易程度不同的编程习题。

本书可以用做计算机科学专业低年级本科生的专业课程的教材，也可以作为软件开发人员学习并行程序设计的专业参考书。

Peter S. Pacheco: An Introduction to Parallel Programming (ISBN 978-0-12-374260-5).

Copyright © 2011 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.

本书简体中文版由机械工业出版社与 Elsevier (Singapore) Pte Ltd. 在中国大陆境内合作出版。本版仅限在中国境内（不包括中国香港特别行政区及中国台湾地区）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。

封底无防伪标均为盗版
版权所有，侵权必究
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：01-2011-4803

图书在版编目（CIP）数据

并行程序设计导论/（美）帕切克（Pacheco，P. S.）著；邓倩妮等译．—北京：机械工业出版社，2012. 8

（计算机科学丛书）

书名原文：An Introduction to Parallel Programming

ISBN 978-7-111-39284-2

Ⅰ. 并…　Ⅱ. ①帕…　②邓…　Ⅲ. 并行程序－程序设计　Ⅳ. TP311. 11

中国版本图书馆 CIP 数据核字（2012）第 173261 号

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：盛思源
北京市荣盛彩色印刷有限公司印刷

2013 年 1 月第 1 版第 1 次印刷
185mm × 260mm · 16. 5 印张
标准书号：ISBN 978-7-111-39284-2
定价：49. 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
购书热线：(010) 68326294；88379649；68995259
投稿热线：(010) 88379604
读者信箱：hzjsj@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，Afred 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

华章科技图书出版中心

译者序

An Introduction to Parallel Programming

本书在对并行硬件和并行软件进行知识总结之后，着重介绍如何利用 MPI、Pthreads 和 OpenMP 开发高效的并行程序。本书的特点在于：

1）文字流畅，易于理解。本书内容通俗易懂，简洁实用。清晰的概念解释，配以丰富的实例和易懂的代码，对帮助初学者理解并行程序设计的基本手段非常重要，可以帮助读者很快掌握设计并行程序的基本方法。

2）循序渐进，由浅及深。本书分别介绍了如何利用 MPI、Pthreads 和 OpenMP 进行并行程序设计。每一章都从最基本的实例开始示范，再介绍一些常见问题不同的实现方法，最后分析和比较不同实现方法的性能。这不仅能帮助初学者快速掌握并行编程方法，还能让读者进一步学习开发高效并行程序设计的方法。

3）重实践，重开发。各章包含了详细介绍的编程实例，以及不同难易程度的编程习题。

本书不仅适合作为计算机专业并行程序设计的课程教材，对需要通过并行程序设计提高计算性能的其他学科（如物理、机械、生物医药等专业）的技术人员，也可以作为参考手册。

本书由上海交通大学邓倩妮副教授主持翻译定稿。此外，冯叶、曾卫、黄鑫、黄叶伟、戴云晶、王强和吕品也参加了本书部分翻译工作，黄鑫还参与了本书的校对工作，对他们的支持和帮助，在此表示衷心的感谢。

由于时间和水平有限，翻译中难免存在不准确，敬请读者指正。

译者

2012 年 6 月

试读结束：需要全本请在线购买： www.ertongbook.com

本书赞誉

随着多核处理器和云计算系统的广泛应用，毫无疑问，并行计算不再是计算世界中被束之高阁的偏门领域。并行性已经成为有效利用资源的首要因素。由 Peter Pacheco（彼得·帕切克）撰写的这本新教材，对于刚开始学术生涯的学生掌握并行计算的艺术和实践很有帮助。

Duncan Buell

南卡罗来纳州大学计算机科学与工程系

本书阐述了两个越来越重要的领域：使用 Pthreads 和 OpenMP 进行共享内存编程，以及使用 MPI 进行分布式内存编程的基本方法。更重要的是，通过指出可能出现的性能错误，强调好的编程实践的重要性。这些主题包含在计算机科学、物理学和数学等多个学科中。各章包含了大量不同难易程度的编程习题。对希望学习并行编程技巧、更新知识面的学生或专业人士来说，本书是一本理想的参考书。

Leigh Little

纽约州立大学布罗科波特学院计算机科学系

本书是一本精心撰写的、全面介绍并行计算的书籍。学生以及相关领域从业人员会从本书的相关最新信息中获益匪浅。Peter Pacheco 通俗易懂的写作手法，结合各种有趣的实例，使本书引人入胜。在并行计算这个瞬息万变、不断发展的领域里，本书深入浅出、全面地介绍了并行软件和并行硬件的方方面面。

Kathy J. Liszka

阿克伦大学计算机科学系

并行计算就是未来！本书通过实用而有益的例子，介绍了这门复杂的学科。

Andrew N. Sloss，FBCS

ARM 公司顾问工程师，《ARM System Developer's Guide》作者

前 言

An Introduction to Parallel Programming

并行硬件已经普及了一段时间。现在已经很难找到一台没有多核处理器的笔记本、台式机或者服务器。在20世纪90年代还是高性能工作站的Beowulf集群，而今已经达到普及程度。与此同时，云计算的出现使得分布式内存系统与台式机一样便于访问。尽管如此，大多数计算机专业的学生在毕业时拥有很少甚至几乎没有任何并行编程经验。虽然许多学院或大学为高年级学生提供并行计算选修课程，但因为计算机科学专业有过多的必修课要求，很多人在毕业时甚至没有写过一个多线程或者多进程的程序。

毫无疑问，这样的现状是需要改变的。虽然许多程序在单核上获得了较满意的性能，但是计算机科学家们必须意识到：并行化有潜力使性能得到巨大提升。当需求提升时，他们应该具备开发这种潜力的能力。

本书旨在部分地解决该问题。它介绍如何使用MPI、Pthreads和OpenMP编写并行程序。MPI、Pthreads和OpenMP是三个广泛应用在并行编程中的应用程序编程接口（Application Programming Interface，API）。本书的预期读者是需要编写并行程序的学生和专业人员。阅读本书仅需要很少的预备知识：大专程度的数学知识和使用C语言编写串行程序的能力。前导知识要求少，因为我们认为学生应该尽快具备编写并行系统的能力。

在旧金山大学，计算机科学专业的学生可以通过学习以本书为教材的课程来达到专业课的要求。“计算机科学导论”是大多数新生在第一个学期学习的课程，本书介绍的课程可以安排为它的后续课程。我们将这门课程作为并行计算的相关课程已经有6年时间。根据我们的经验，学生完全不需要从中、高年级才开始编写并行程序。相反，这门课程十分受欢迎。通过学习这门导论课，学生可以很轻松地将并行性应用于其他课程。

第二学期的新生可以通过课堂学习编写并行程序，而带着目的进行学习的计算机专业人员可以自学并行编程。我们希望本书对于他们是有用的资源。

关于本书

正如前面所说的，本书的主要目的是：让那些对计算机科学只有有限背景知识、没有并行性经验的读者学习使用MPI、Pthreads和OpenMP进行并行编程。为了让本书使用起来更灵活，我们尽量让那些对API没有兴趣的读者花费较少的时间就能很容易地阅读剩下的部分。因此，针对这三个API的章节是相互独立的：可以按任意顺序阅读它们，甚至可以跳过一两个章节。但是，这种独立性有一定代价：有些内容会在这些章节中重复提到。当然，重复的内容可以简单浏览或者直接跳过。

没有并行计算经验的读者应该先阅读第1章。第1章尝试用相关的非技术性的语言解释为什么并行系统已经在计算机领域中占有重要地位。这一章还为并行系统和并行编程提供了一个简短的介绍。

第2章介绍计算机硬件和软件的一些技术背景。在API章节开始之前，许多关于硬件的材料可以先粗略浏览。第3章、第4章、第5章分别介绍使用MPI、Pthreads、OpenMP进行编程。

在第6章中，我们开发了两个更长的程序：并行n体问题求解和并行树搜索。这两个程序都使用上述的三个API。第7章提供一个简单的列表，给出并行计算各个方面的补充信息。

我们使用C语言来开发程序，因为这三个API都使用C语言接口，同时也因为C语言是相对简单易学的语言，尤其是对于C++和Java程序员来说，他们已经对C的控制结构非常熟悉。

课堂使用

本书来源于旧金山大学低年级本科生的课程。这门课满足了计算机科学的专业课要求，同时也是本科生操作系统课程的先修课。本课程唯一的要求是在第一学期的“计算机科学导论”课程中获得 B 及以上成绩，或在第二学期的“计算机科学导论”课程中获得 C 及以上成绩。课程的前四个星期介绍 C 语言编程。由于大部分学生已经编写过 Java 程序，因此课程主要集中在 C 语言中如何使用指针上[⊖]。课程剩下的部分介绍使用 MPI、Pthreads、OpenMP 编程。

上述的大部分内容集中在本书第 1 章、第 3 章、第 4 章、第 5 章中，并在第 2 章和第 6 章中有少量提及。当需求提高时，第 2 章的背景知识应该介绍。比如，在讨论 OpenMP（第 5 章）缓存一致性问题前，应该先介绍第 2 章中缓存的知识。

本课程的作业包括每周作业、5 个编程作业、两次期中考试和一次期末考试。作业中经常涉及编写一个简短的程序或者对现有的程序进行一些小的改进。这些作业的目的是保证学生能够跟上课程，并且根据课堂上的想法得到实际操作经验。这些作业的布置是本课程成功的重要原因。课本中大多数习题与这些简短的作业是相适应的。

编程作业中需要编写更大的程序，但我们一般会为学生提供非常多的指导：我们经常在作业中提供伪代码，并在课堂上讨论较难的部分。这些额外的指导是非常有效的：那些需要花费学生大量时间完成的编程作业变得不再难以提交。期中和期末考试的结果，以及讲授操作系统课程的教师的报告都表明，这门课程对于教授学生如何编写并行程序是非常成功的。

对于其他高级并行计算课程，本书和它的在线辅助材料可以作为补充，许多关于这三个 API 语法和语义的信息都可以作为课外阅读资料。本书也可以作为工程方面的课程或者与计算机科学无关但涉及并行计算的课程的补充材料。

辅助材料

关于本书的勘误表和一些相关材料，请访问本书出版社的网站（http://www.elsevierdirect.com/），那里还可以下载完整的课件、图表、习题答案和编程作业。所有的用户都可以下载课本中讨论过的较长程序。

我们非常感谢读者提出任何发现的错误。如果你发现错误，请发送邮件至 peter@usfca.edu。

⊖ 有趣的是，很多学生认为 C 语言中的指针比 MPI 编程更困难。

致　谢

在编写本书的过程中，得到了许多人的帮助。非常感谢那些在本书编写初期阅读并提出建议的评阅人：Fikret Ercal（Missouri University of Science and Technology），Dan Harvey（Southern Oregon University），Joel Hollingsworth（Elon University），Jens Mache（Lewis and Clark College），Don McLaughlin（West Virginia University），Manish Parashar（Rutgers University），Charlie Peck（Earlham College），Stephen C. Renk（North Central College），Rolfe Josef Sassenfeld（The University of Texas at El Paso），Joseph Sloan（Wofford College），Michela Taufer（University of Delaware），Pearl Wang（George Mason University），Bob Weems（University of Texas at Arlington），Cheng-Zhong Xu（Wayne State University）。

我同样非常感谢以下评阅人，他们针对各个章节提出了不同的建议：Duncan Buell（University of South Carolina），Matthias Gobbert（University of Maryland，Baltimore County），Krishna Kavi（University of North Texas），Hong Lin（University of Houston-Downtown），Kathy Liszka（University of Akron），Leigh Little（The State University of New York），Xinlian Liu（Hood College），Henry Tufo（University of Colorado at Boulder），Andrew Sloss（Consultant Engineer，ARM），Gengbin Zheng（University of Illinois）。他们的意见和建议使得本书得到巨大的改进。当然，我个人需要对依然存在的错误和漏洞负责。

Kathy Liszka 为采用本书的教师准备了幻灯片；Jinyoung Choi，我先前的学生，为本书准备了一份答案手册。同样感谢他们。

Morgan Kaufmann 的员工在整个项目中对我提供了非常多的帮助。我尤其感谢编辑 Nate McFadden，他给了我很多宝贵的建议，并做了了不起的工作安排，在过去几年中，他与我讨论了所有遇到的问题，并表现了极大的耐心。同时感谢 Marily Rash 和 Megan Guiney，他们在编写过程中表现得非常迅速有效。

旧金山大学计算机科学和数学系的同事们在编写本书的过程中为我提供了非常多的帮助。这里特别感谢 Gregory Benson 教授：他对于并行计算的理解（尤其是 Pthreads 和信号量）是我宝贵的资源。我也非常感谢我们的系统管理员 Alexey Fedosov 和 Colin Bean，在我编写本书的过程中，他们耐心并有效地处理了出现的所有“紧急情况”。

如果没有我的朋友 Holly Cohn、John Dean、Robert Miller 的鼓励和道义上的支持，我将永远不可能完成本书，他们帮我度过了一段非常困难的时期，我永远感谢他们。

我最大的感谢致予我的学生，是他们向我展示了什么是过于简单，什么是过于困难。总之，他们教会了我如何去教授并行计算。我向他们致以最深切的感谢。

出版者的话
译者序
本书赞誉
前言
致谢

第1章 为什么要并行计算 …… 1
1.1 为什么需要不断提升的性能 …… 1
1.2 为什么需要构建并行系统 …… 2
1.3 为什么需要编写并行程序 …… 2
1.4 怎样编写并行程序 …… 4
1.5 我们将做什么 …… 5
1.6 并发、并行、分布式 …… 6
1.7 本书的其余部分 …… 7
1.8 警告 …… 7
1.9 字体约定 …… 7
1.10 小结 …… 8
1.11 习题 …… 8

第2章 并行硬件和并行软件 …… 10
2.1 背景知识 …… 10
2.1.1 冯·诺依曼结构 …… 10
2.1.2 进程、多任务及线程 …… 11
2.2 对冯·诺依曼模型的改进 …… 12
2.2.1 Cache基础知识 …… 12
2.2.2 Cache映射 …… 13
2.2.3 Cache和程序：一个实例 …… 14
2.2.4 虚拟存储器 …… 15
2.2.5 指令级并行 …… 17
2.2.6 硬件多线程 …… 19
2.3 并行硬件 …… 19
2.3.1 SIMD系统 …… 20
2.3.2 MIMD系统 …… 22
2.3.3 互连网络 …… 23
2.3.4 Cache一致性 …… 28
2.3.5 共享内存与分布式内存 …… 30
2.4 并行软件 …… 31
2.4.1 注意事项 …… 31
2.4.2 进程或线程的协调 …… 31
2.4.3 共享内存 …… 32
2.4.4 分布式内存 …… 35
2.4.5 混合系统编程 …… 37
2.5 输入和输出 …… 37
2.6 性能 …… 38
2.6.1 加速比和效率 …… 38
2.6.2 阿姆达尔定律 …… 40
2.6.3 可扩展性 …… 41
2.6.4 计时 …… 41
2.7 并行程序设计 …… 43
2.8 编写和运行并行程序 …… 46
2.9 假设 …… 47
2.10 小结 …… 47
2.10.1 串行系统 …… 47
2.10.2 并行硬件 …… 48
2.10.3 并行软件 …… 49
2.10.4 输入和输出 …… 50
2.10.5 性能 …… 50
2.10.6 并行程序设计 …… 51
2.10.7 假设 …… 51
2.11 习题 …… 51

第3章　用 MPI 进行分布式内存编程 …… 54
3.1 预备知识 …… 54
3.1.1 编译与执行 …… 55
3.1.2 MPI 程序 …… 56
3.1.3 MPI_Init 和 MPI_Finalize …… 56
3.1.4 通信子、MPI_Comm_size 和 MPI_Comm_rank …… 57
3.1.5 SPMD 程序 …… 57
3.1.6 通信 …… 57
3.1.7 MPI_Send …… 58
3.1.8 MPI_Recv …… 59
3.1.9 消息匹配 …… 59
3.1.10 status_p 参数 …… 60
3.1.11 MPI_Send 和 MPI_Recv 的语义 …… 61
3.1.12 潜在的陷阱 …… 61
3.2 用 MPI 来实现梯形积分法 …… 62
3.2.1 梯形积分法 …… 62
3.2.2 并行化梯形积分法 …… 62
3.3 I/O 处理 …… 64
3.3.1 输出 …… 64
3.3.2 输入 …… 66
3.4 集合通信 …… 67
3.4.1 树形结构通信 …… 67
3.4.2 MPI_Reduce …… 68
3.4.3 集合通信与点对点通信 …… 69
3.4.4 MPI_Allreduce …… 70
3.4.5 广播 …… 70
3.4.6 数据分发 …… 72
3.4.7 散射 …… 73
3.4.8 聚集 …… 74
3.4.9 全局聚集 …… 75
3.5 MPI 的派生数据类型 …… 77
3.6 MPI 程序的性能评估 …… 79
3.6.1 计时 …… 79
3.6.2 结果 …… 82
3.6.3 加速比和效率 …… 83
3.6.4 可扩展性 …… 84
3.7 并行排序算法 …… 85
3.7.1 简单的串行排序算法 …… 85
3.7.2 并行奇偶交换排序 …… 86
3.7.3 MPI 程序的安全性 …… 88
3.7.4 并行奇偶交换排序算法的重要内容 …… 90
3.8 小结 …… 91
3.9 习题 …… 93
3.10 编程作业 …… 98

第4章　用 Pthreads 进行共享内存编程 …… 100
4.1 进程、线程和 Pthreads …… 100
4.2 “Hello，World” 程序 …… 101
4.2.1 执行 …… 101
4.2.2 准备工作 …… 102
4.2.3 启动线程 …… 103
4.2.4 运行线程 …… 104
4.2.5 停止线程 …… 105
4.2.6 错误检查 …… 105
4.2.7 启动线程的其他方法 …… 105
4.3 矩阵－向量乘法 …… 106
4.4 临界区 …… 107
4.5 忙等待 …… 109
4.6 互斥量 …… 112
4.7 生产者－消费者同步和信号量 …… 114
4.8 路障和条件变量 …… 117
4.8.1 忙等待和互斥量 …… 117
4.8.2 信号量 …… 118
4.8.3 条件变量 …… 119
4.8.4 Pthreads 路障 …… 121
4.9 读写锁 …… 121

试读结束：需要全本请在线购买： www.ertongbook.com

4.9.1 链表函数 …… 121
4.9.2 多线程链表 …… 122
4.9.3 Pthreads 读写锁 …… 124
4.9.4 不同实现方案的性能 …… 125
4.9.5 实现读写锁 …… 126
4.10 缓存、缓存一致性和伪共享 …… 127
4.11 线程安全性 …… 130
4.12 小结 …… 132
4.13 习题 …… 134
4.14 编程作业 …… 137

第5章 用 OpenMP 进行共享内存编程 …… 139

5.1 预备知识 …… 140
5.1.1 编译和运行 OpenMP 程序 …… 140
5.1.2 程序 …… 141
5.1.3 错误检查 …… 143
5.2 梯形积分法 …… 143
5.3 变量的作用域 …… 147
5.4 归约子句 …… 147
5.5 parallel for 指令 …… 150
5.5.1 警告 …… 150
5.5.2 数据依赖性 …… 151
5.5.3 寻找循环依赖 …… 152
5.5.4 π 值估计 …… 153
5.5.5 关于作用域的更多问题 …… 154
5.6 更多关于 OpenMP 的循环：排序 …… 155
5.6.1 冒泡排序 …… 155
5.6.2 奇偶变换排序 …… 156
5.7 循环调度 …… 158
5.7.1 schedule 子句 …… 159
5.7.2 static 调度类型 …… 159
5.7.3 dynamic 和 guided 调度类型 …… 160
5.7.4 runtime 调度类型 …… 160
5.7.5 调度选择 …… 161
5.8 生产者和消费者问题 …… 162
5.8.1 队列 …… 162
5.8.2 消息传递 …… 162
5.8.3 发送消息 …… 162
5.8.4 接收消息 …… 163
5.8.5 终止检测 …… 163
5.8.6 启动 …… 164
5.8.7 atomic 指令 …… 164
5.8.8 临界区和锁 …… 165
5.8.9 在消息传递程序中使用锁 …… 166
5.8.10 critical 指令、atomic 指令、锁的比较 …… 167
5.8.11 经验 …… 167
5.9 缓存、缓存一致性、伪共享 …… 168
5.10 线程安全性 …… 172
5.11 小结 …… 174
5.12 习题 …… 176
5.13 编程作业 …… 179

第6章 并行程序开发 …… 182

6.1 n 体问题的两种解决方法 …… 182
6.1.1 问题 …… 182
6.1.2 两个串行程序 …… 183
6.1.3 并行化 n 体算法 …… 186
6.1.4 关于 I/O …… 188
6.1.5 用 OpenMP 并行化基本算法 …… 188
6.1.6 用 OpenMP 并行化简化算法 …… 191
6.1.7 评估 OpenMP 程序 …… 193
6.1.8 用 Pthreads 并行化算法 …… 194
6.1.9 用 MPI 并行化基本算法 …… 195
6.1.10 用 MPI 并行化简化算法 …… 197

6.1.11 MPI 程序的性能 …… 200
6.2 树形搜索 …… 201
6.2.1 递归的深度优先搜索 …… 203
6.2.2 非递归的深度优先搜索 …… 204
6.2.3 串行实现所用的数据结构 …… 205
6.2.4 串行实现的性能 …… 206
6.2.5 树形搜索的并行化 …… 206
6.2.6 采用 Pthreads 实现的静态并行化树搜索 …… 208
6.2.7 采用 Pthreads 实现的动态并行化树搜索 …… 209
6.2.8 Pthreads 树搜索程序的评估 …… 212
6.2.9 采用 OpenMp 实现的并行化树搜索程序 …… 213
6.2.10 OpenMp 实现的性能 …… 215
6.2.11 采用 MPI 和静态划分来实现树搜索 …… 215
6.2.12 采用 MPI 和动态划分来实现树搜索 …… 221
6.3 忠告 …… 226
6.4 选择哪个 API …… 226
6.5 小结 …… 227
6.5.1 Pthreads 和 OpenMP …… 228
6.5.2 MPI …… 228
6.6 习题 …… 230
6.7 编程作业 …… 236
第7章 接下来的学习方向 …… 238
参考文献 …… 240
索引 …… 242

为什么要并行计算

从1986年到2002年，微处理器的性能以平均每年50%的速度不断提升［27］。这样史无前例的性能提升，使得用户和软件开发人员只需要等待下一代微处理器的出现，就能够获得应用程序的性能提升。但是，从2002年开始，单处理器的性能提升速度降低到每年大约20%，这个差异是巨大的：如果以每年50%的速度提升，在10年里微处理器的性能会提升60倍，而以20%的速度，10年里只能提升6倍。

此外，性能提升速度的差异也极大地改变了处理器的设计。到2005年，大部分主流的微处理器制造商已决定通过并行处理来快速提升微处理器的性能。他们不再继续开发速度更快的单处理器芯片，而是开始将多个完整的单处理器放到一个集成电路芯片上。

这一变化对软件开发人员带来了重大的影响：大多数**串行**程序是在单个处理器上运行的，不会因为简单地增加更多的处理器就获得极大的性能提高。串行程序不会意识到多个处理器的存在，它们在一个多处理器系统上运行的性能，往往与在多处理器系统的一个处理器上运行的性能相同。

所有这一切引出如下问题：

1）为什么我们要关心并行？单处理器系统不是已经足够快了吗？毕竟每年20%的性能提升也是很可观的。

2）为什么微处理器制造商不能继续研制更快的单处理器系统？为什么要研制**并行系统**？为什么要研制多处理器系统？

3）为什么不能编写程序，将串行程序自动转换成可以充分利用多处理器的**并行程序**？

下面我们简单地回答上述问题。但请记住：有些问题的答案不是一成不变的。例如，每年20%的性能提升对大多数应用程序来说是绰绰有余的了。

1.1 为什么需要不断提升的性能

过去几十年中，不断提升的计算能力已经成为许多飞速发展领域（如科学、互联网、娱乐等）的核心力量。例如：人类基因解码、更准确的医疗成像、更快速精确的网络搜索、更真实的电脑游戏，都离不开计算能力的提高。确实，没有早期的提高，现在很多应用的计算能力提升将会很难实现，甚至不可能实现。但是，我们不能满足于现状。随着计算能力的提升，我们要考虑解决的问题也在增加，如下就是一些例子：

- *气候模拟*：为了更好地理解气候变化，我们需要更加精确的计算模型，这种模型必须包括大气、海洋、陆地以及极地冰川之间的相互关系。我们需要对各种因素如何影响全球气候做详细研究。
- *蛋白质折叠*：人们相信错误折叠的蛋白质与亨廷顿病、帕金森病、老年痴呆症等疾病有千丝万缕的联系，但现有的计算性能严重限制了研究复杂分子（如蛋白质）结构的能力。
- *药物发现*：不断提高的计算能力可以从不同方面促进新的医学研究。例如，有许多药物只是对一小部分患者有效。我们可以通过仔细分析疗效欠佳患者的基因来找到替代的药物，但这需要大规模的基因组计算和分析。

- *能源研究*：不断提高的计算能力可以为某些技术（如风力涡轮机、太阳能电池和蓄电池）构建更详细的模型。这些模型能够为建立更高效清洁的能源提供信息。
- *数据分析*：每天都会产生大量的数据。据估计，全球范围存储的数据每两年翻一番［28］，而这些数据中的大部分在未经分析前是无用的。例如，了解人类 DNA 的核苷酸序列本身是没有用的，而理解这个序列如何影响生长发育，以及它是如何引起疾病的，需要大规模的数据分析。除了基因组学，欧洲核子研究中心（CERN）的大型强子对撞机、医疗成像、天文研究、网络搜索引擎也会产生海量的数据，并需要对这些数据进行大规模分析。

2 上述这些问题以及其他问题的解决都需要更强大的计算能力。

1.2 为什么需要构建并行系统

单处理器性能大幅度提升的主要原因之一，是日益增加的集成电路晶体管密度（晶体管是电子开关）。随着晶体管尺寸的减小，晶体管的传递速度增快，集成电路整体的速度也增快。但是，随着晶体管速度的增快，它们的能耗也相应增加。大多数能量是以热能的形式消耗，当一块集成电路变得太热的时候，就会变得不可靠。在 21 世纪的第一个 10 年中，用空气冷却的集成电路的散热能力已经达到了极限［26］。

因此，通过继续增快集成电路的速度来提高处理器性能的方法变得不再可行。但是集成电路晶体管的密度还在增加，并且还会持续一段时间。而且，既然计算方法对改善我们现有的方式有潜在的推动作用，那么继续发掘更强的计算能力就是必要而迫切的。最后，如果集成电路制造商不能继续推出更新、更好的产品，那么它很快就会被淘汰。

我们如何利用还在不断增加的晶体管密度？答案是*并行*。集成电路制造商的决策是：与其构建更快、更复杂的单处理器，不如在单个芯片上放置多个相对简单的处理器。这样的集成电路称为**多核**处理器。**核**已经成为中央处理器或者 CPU 的代名词。在这样的设定下，传统的只有一个 CPU 的处理器称为**单核**系统。

1.3 为什么需要编写并行程序

大多数为传统单核系统编写的程序无法利用多核处理器。虽然可以在多核系统上运行一个程序的多个实例，但这样意义不大。例如，在多个处理器上运行一个喜爱的游戏程序的多个实例并不是我们需要的。我们需要的是这个程序能够更快地运行，有更加逼真的图像。为了达到这一目的，就需要将串行程序改写为并行程序，或者编写一个翻译程序来自动地将串行程序翻译成并行程序，只有这样才能充分利用多核。不幸的是，研究人员在自动将串行程序（例如 C 或C++编写的程序）转换成并行程序上鲜有突破。

这并不令人惊讶。尽管我们可以编写一些程序，让这些程序辨识串行程序的常见结构，并自动将这些结构转换成并行程序的结构，但转化后的并行程序在实际运行时可能很低效。例如，两个 $n \times n$ 的矩阵相乘是由多个点积操作组成的一个序列，但是将矩阵相乘并行化为一系列并行执

3 行的点积操作，在很多系统上运行效率并不高。

一个串行程序的高效并行实现可能不是通过发掘其中每一个步骤的高效并行实现来获得，相反，最好的并行化实现可能是通过一步步回溯，然后发现一个全新的算法来获得的。

举例来说，假设我们需要计算 n 个数的值再累加求和，如下是串行代码：

```
sum = 0;
for (i = 0; i < n; i++) {
   x = Compute_next_value(. . .);
   sum += x;
}
```

现在我们假设有 p 个核，且 p 远小于 n，那么每个核能够计算大约 n/p 个数的值并累加求和，以得到部分和：

```
my_sum = 0;
my_first_i = . . . ;
my_last_i = . . . ;
for (my_i = my_first_i; my_i < my_last_i; my_i++) {
   my_x = Compute_next_value(. . .);
   my_sum += my_x;
}
```

此处的前缀 my_代表每个核都使用自己的私有变量，并且每个核能够独立于其他核来执行该代码块。

每个核都执行完代码后，变量 my_sum 中就会存储调用 Compute_next_value 获得的值的和。例如，假如有 8 个核，$n=24$，24 次调用 Compute_next_value 获得如下的值：

1，4，3　9，2，8　5，1，1　6，2，7　2，5，0　4，1，8　6，5，1　2，3，9

这样存储在 my_sum 中的值将是：

核	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

这里，我们假定用非负整数 0，1，…，$p-1$ 来标识各个核，p 是核的总数。

当各个核都计算完各自的 my_sum 值后，将自己的结果值发送给一个指定为“master”的核（主核），master 核将收到的部分和累加而得到全局总和：

```
if (I'm the master core) {
   sum = my_x;
   for each core other than myself {
      receive value from core;
      sum += value;
   }
}else{
  send my_x to the master;
}
```

4

在我们的例子中，假如 master 核是 0 号核，它将部分和累加求全局总和 8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95。

但是，可能你已经想到一个更好的方法了——特别是当核的数目比较多的时候，不再由 master 核计算所有部分和的累加工作，可以将各个核两两结对，0 号核将自己的部分和与 1 号核的部分和做加法，2 号核将自己的部分和与 3 号核的部分和做加法，4 号核将自己的部分和与 5 号核的部分和做加法，以次类推。然后，再在偶数核上重复累加部分和：0 号核加上 2 号核，4 号核加上 6 号核，以次类推，如图 1-1 所示。圆圈表示当前核所得到的和，箭头表示一个核将自己的部分和发送给另一个核，加号表示一个核在收到另一个核发送来的部分和后与自己本身的部分和相加。

上述两种计算全局总和的算法中，master 核（0 号核）承担了更多的工作量，整个程序计算全局总和的时间就等于 master 核的计算时间。在 8 个核的情况下，第一种方法中，master 核需要执行 7 次接收操作，而第二种方法中，master 核仅需要执行 3 次接收操作。因此第二种方法比第一种方法快 2 倍，当有更多的核时，两者的差异更大。在 1000 个核的情况下，第一种方法需要 999 次接收和加法操作，而第二种方法只需要 10 次，提高了 100 倍。5

非常明显，第一种计算全局总和的方法是对串行求和程序的一般化：将求和的工作在核之间平分，等到每个核都计算出部分和之后，master 简单地重复串行程序中基本的串行求和。如果有

p 个核，master 就需要计算 p 个值的总和。而第二种计算全局总和的方法与原来的串行程序没有多大关系。

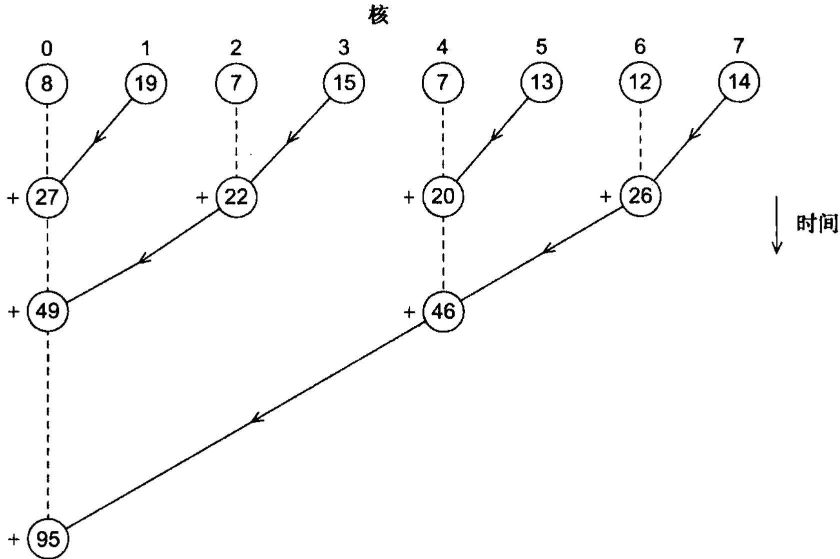


图 1-1　多个核共同计算形成一个全局总和

问题的关键是：除非事先已经定义好这么一个高效的求全局和算法，否则翻译程序不太可能发现第二种计算全局总和的方法。在翻译的时候，通过识别出原来的串行循环，将其替换为预定义好的高效并行求和算法。

我们期望通过编写软件，识别出常见的串行结构，并对其进行有效的**并行化**，使其能够利用多个核。但是，当我们将此原则应用于更复杂的串行程序时，识别结构将变得越来越困难，转换为事先定义好的高效并行化方法也变得越来越不可能。

因此，我们不能再继续简单地编写串行程序，我们必须编写并行程序来发掘多核处理器的潜在性能。

1.4　怎样编写并行程序

对于这个问题，有多种可能的解决方案。大部分方案的基本思想都是将要完成的任务分配给各个核。有两种广泛采用的方法：**任务并行**和**数据并行**。任务并行是指将待解决问题所需要执行的各个任务分配到各个核上执行。而数据并行是指将待解决问题所需要处理的数据分配给各个核，每个核在分配到的数据集上执行大致相似的操作。

举例来说，假如 P 教授进行“英国文学调查”的授课，她有 100 个学生，还有 4 个助教（Teaching Assistant，TA），A 先生、B 女士、C 先生、D 女士。学期结束的时候，要进行一次期末测试，这个测试中包括 5 道题。为了给学生打分，P 教授和她的助教可能有如下两种批改方案：每人负责给一个问题打分；或者将学生分成 5 组，每人负责一组，即 20 个学生。

在这两种方案中，P 教授和她的助教充当核的角色。第一种方案可以认为是任务并行的例子。有 5 个任务需要执行，即给第一个问题打分，给第二个问题打分……给第五个问题打分。当然，每个打分人审阅的题目是不一样的，比如：第一个问题是关于莎士比亚的，第二个问题可能是关于米利托的，所以 P 教授和她的助教是在“执行不同的指令”。

6

第二种方案可以认为是数据并行的例子。“数据”是学生的卷子，在不同的核（打分人）之间平分，每个核执行大致相似的打分“指令”。

试读结束：需要全本请在线购买： www.ertongbook.com