TURING 图灵程序设计丛书

PEARSON

Multicore Application Programming

For Windows, Linux, and Oracle® Solaris

多核应用编程实战

- 面向多种平台的丰富示例
- 脱离特定语言的多核编程思想
- 不限于某一并行处理方法的全面分析

【美】Darryl Gove 著　郭晴霞 译

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Multicore Application Programming

For Windows, Linux, and Oracle® Solaris

多核应用编程实战

【美】 Darryl Gove 著

郭晴霞 译

人民邮电出版社

图书在版编目（CIP）数据

多核应用编程实战 /（美）戈夫（Gove,D.）著；郭晴霞译. -- 北京：人民邮电出版社，2013.6
（图灵程序设计丛书）
书名原文：Multicore application programming:for windows,linux,and Oracle Solaris
ISBN 978-7-115-31750-6

Ⅰ. ①多… Ⅱ. ①戈… ②郭… Ⅲ. ①并行程序－程序设计 Ⅳ. ①TP311.11

中国版本图书馆CIP数据核字(2013)第091677号

内 容 提 要

本书是一本全面实用的多核应用编程指南，旨在介绍如何编写功能正确、性能优越且适合扩展为在多个CPU核心的系统运行的应用程序。本书面向多种操作系统和处理器类型引用程序示例，内容涵盖类UNIX操作系统（Linux、Oracle Solaris、OS X）和Windows系统上多核应用的编写方法、多核的硬件实现对应用程序的性能影响、编写并行应用程序时要避免的潜在问题，以及如何编写可扩展至大量并行线程的应用程序。

本书适合所有C程序员学习参考。

图灵程序设计丛书
多核应用编程实战

◆ 著 [美] Darryl Gove
译 郭晴霞
责任编辑 毛倩倩
责任印制 焦志炜

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：800×1000 1/16
印张：22.25
字数：526千字 2013年6月第1版
印数：1-4 000册 2013年6月河北第1次印刷

著作权合同登记号 图字：01-2011-1851号
ISBN 978-7-115-31750-6

定价：79.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

多年以来，家用电脑总是给人以同时处理多任务的错觉。这是因为处理器每秒钟能在不同的运行任务之间切换很多次，于是给人多个任务同时运行的表象，但也仅仅是表象。实际上，当计算机处理某个任务时，其他任务不会有任何进展。对于一次只能执行一个任务的电脑，我们可以说它只有一个处理器、CPU（“核心”）。核心是处理器中承担实际工作的部分。

近来，即使是家用电脑也配备了多核处理器。现在已经很难买到非多核计算机了。在多核机上，每个核心执行一个任务，从而真正做到了同时处理多任务。

要理解多核的意义，最好假设要通过一台电脑把摄像机里的电影转换为适合DVD刻录的格式。这个过程需要大量计算（密集地读写磁盘），而解压输入视频和将之转换为要刻录成盘的压缩输出视频会占用处理器的绝大部分时间。

如果忽略磁盘和内存的问题，那么也许可以在单核系统上同时转换两部电影。这两个任务可同时开始，但处理器会先用一段时间转换一个视频，然后再用一段时间转换另一个。因为处理器同一时刻只能执行一个任务，所以实际只有一个视频被压缩。如果用进度条显示转换过程的话，那么可以看到两个进度条都朝着100%完成的方向前进，但实际上转换两个视频所用的时间约为转换一个视频的两倍。

多核系统则不同，进行视频转换的核心是两个或更多，而每个核心都可处理一项任务。因此，让系统在同一时间处理两部电影将使用两个核心，且转换时间与转换一部电影时相同，即同样时间内可以完成两倍的工作量。

多核系统能在单位时间内完成更多工作，比如用单核系统转换一部电影的时间转换两部电影。此外，多核系统也能用不同的方式来分解工作。例如，多个核心可以共同转换同一部电影。在这种情况下，双核系统转换一部电影比单核系统快一倍。

本书讲解如何利用多核系统进行开发。人们常说这个话题复杂或难以理解，某种意义上这么说没有错。与任何一种编程技术一样，多核编程很难同时做到正确和高性能。但另一方面，利用多核系统显著提升应用程序性能或提高单位时间内所完成工作量的方式也有很多，只是实现的难易程度不同。

也许说“多核编程很容易”过于乐观，比较切合实际的说法是：多核编程不见得比从结构化编程转向面向对象编程更难。本书将帮助你了解编写多核系统的应用程序涉及的难点，使你能写出功能正确、性能优越，且适合扩展为在多个CPU核心运行的应用程序。

读者对象

既然已经读到这里，那么你很可能就是本书的目标读者。本书是一本实用指南，指导你编写能充分利用多核系统优势的应用程序。本书并不专门描述并行处理的某个具体方法，而是涵盖了各种方法。本书也并未拘泥于某个特定的平台，而是给出了多种操作系统和处理器的程序示例。本书还涵盖了某些高级主题，但都给出了足够的背景知识，保证读者能够理解。

本书是为熟悉C语言并有相当编程能力的读者而作。本书的目的并非教授编程语言，而是从更高层次考量如何编写功能正确、性能优良、可扩展为在多个CPU核心运行的代码。

本书的示例使用了SPARC或x86汇编语言。示例简单、有清晰的注释，重点突出，读者无需熟悉汇编语言就能看明白。

本书目标

阅读本书之后，读者将了解为类UNIX操作系统（Linux、Oracle Solaris、OS X）和Windows系统编写多核程序的方法，理解多核的硬件实现对应用程序的影响（好坏都有），知晓编写并行应用程序时的注意事项，并弄清如何编写可扩展为大量并行线程的应用程序。

本书内容

以下是本书各章的简介。

第1章介绍将涉及的硬件和软件概念，概述了处理器的内部结构。对于读者来说，要编写利用多核系统的程序不一定非得了解硬件的工作原理，但理解了处理器架构的基础知识更容易理解后面有关应用程序正确性、性能和扩展的概念。这一章还讨论了线程和进程的概念。

第2章讨论应用程序分析和优化。在把精力花在将应用程序修改为利用多核的程序之前，了解应用程序目前将时间用在哪里至关重要。这一章介绍了应用程序开发周期中的主要性能因素，并探讨如何才能提升性能。

第3章介绍如何利用多核系统在单位时间内执行更多工作，或者说如何减少完成一个任务所需的时间。首先介绍虚拟化，因为虚拟化可以在一个系统中模拟多个系统，且无需修改软件。重点在于多核系统代表了一个机会，让我们无需修改软件即可改变其工作方式。然后，这一章描述可用于编写并行应用程序的各种模式，并讨论这些模式适用的情况。

第4章阐述如何在多个线程之间安全地共享数据。这一章以讨论数据争用开篇，它在多线程代码中最容易导致问题。这一章特别在抽象层面上详述了安全共享数据和同步线程的方法，后续几章针对具体操作系统再详述细节。

第5章介绍如何使用POSIX线程编写并行应用程序。POSIX线程是在类UNIX操作系统（如Linux、Mac OS X和Solaris）上实现的标准。POSIX线程库提供了许多对于编写并行应用程序很实用的构建模块，为开发提供了极大的灵活性和便利。

第6章介绍如何使用Windows本地线程为Windows编写并行应用程序。Windows提供了与POSIX类似的同步和数据共享原语，但Windows和POSIX对这些功能的接口和要求不同。

第7章描述编译器提供的自动并行化机会和限制。这一章还介绍了OpenMP规范，这一规范使编写利用多核处理器的应用程序变得相对简单。

第8章探讨如何抛开操作系统或编译器提供的库功能来编写并行应用程序。我们有充分的理由为同步或共享数据编写自定义代码，比如想要获得更好的控制或更出色的性能。但要编写正常运行的代码，也有不少需要避免的陷阱。

第9章讨论如何对应用程序加以改进以提高其扩展性，从而使多核系统完成的工作尽可能多。这一章介绍了限制扩展性的常见领域，阐述了确定扩展性限制范围的方式。面向多核系统进行开发和面向多处理器系统进行开发的不同之处恰恰在于扩展，而这一章还讨论了硬件实现带来的这一重要差别。

第10章涵盖编写并行应用程序的多种方式。随着多核处理器成为主流，人们也在尝试其他方法，以便克服障碍编写出正确、快速、可扩展的并行代码。

第11章是最后一章，对全书内容进行了总结。

致　谢

这本书得以面世得益于许多人的努力，他们探讨了本书中涵盖的各种问题，审阅了本书并保证其正确性和连贯性。感谢Miriam Blatt、Steve Clamage、Mat Colgrove、Duncan Coutts、Harry Foxwell、Karsten Guthridge、David Lindt、Jim Mauro、Xavier Palathingal、Rob Penland、Steve Schalkhauser、Sukhdeep Sidhu、Peter Strazdins、Ruud van der Pas和Rick Weisner对各章草稿进行审校，他们提供了宝贵的反馈意见。特别感谢向我提供大量详细反馈意见的Richard Friedman。

同时，感谢包括Greg Doench、Michelle Housley、Anna Popick和Michael Thurston在内的Addison-Wesley出版团队，以及兼职的文字编辑Kim Wimpsett，感谢他们对本书的指导、编辑、校对、建议和支持。

我也要借此感谢家人和朋友在我撰写本书期间给予帮助和鼓励。如果没有全体亲友的支持和理解，我绝不可能有时间写作。此外，有人关心写作的进展也给了我莫大的鼓舞。特别感谢我的父母Tony和Maggie以及我的岳父母Geoff和Lucy的热情和支持。

最后，也是最重要的，感谢我的妻子Jenny、我们的儿子Aaron和Timothy以及女儿Emma，是他们极大的热忱和支持给予我动力，激发我探究事物原理并向读者传授这些知识。

版权声明

目　录

第1章 硬件、进程和线程

要编写串行或并行应用程序，并非一定要了解硬件的工作原理，编写代码时完全可以将计算机内部结构视作黑盒子。然而，对处理器内部结构作一个基本了解会使稍后的主题更直观易懂。串行（或单线程）应用程序和并行（或多线程）应用程序之间的关键区别在于，由于多线程的存在，更多系统属性变得对应用程序重要起来。例如，单线程应用程序不会出现多个线程争用相同资源的情况，但对多线程应用程序来说这种情况经常发生。资源可能是缓存空间、内存带宽，或者仅仅是物理内存空间，在这些情况下，硬件特性会反映在应用程序行为的变化上。若对硬件的工作方式有所了解，读者会更容易理解、诊断和纠正应用程序的任何异常行为。

1.1 计算机的内部结构

从根本上讲，一台计算机包括一个或多个处理器以及一些内存，它们通过一些芯片和电线连接起来。此外，还有磁盘驱动器或网卡等辅助设备。

图1-1显示了一台个人计算机的内部结构。计算机由数个元件组成，处理器和内存插在称为*主板*的电路板上，从主板上伸出的电线与磁盘驱动器、DVD光驱等辅助设备相连。视频、网络支持等功能则集成到主板上或通过插卡提供。

如以图表示信息，可能更容易让人了解系统各元件的关联，参见图1-2。在此示意图中，我们将系统的计算部分与辅助设备分开表示。

系统的计算性能主要由处理器和内存的性能决定，而计算性能又将决定计算机执行指令的速度快慢。

辅助设备的性能不是我们所关注的，因为辅助设备的性能比内存和处理器低得多。处理器一秒内传送到内存的数据量以千兆字节为单位，每秒传送到磁盘的数据量则多以兆字节为单位计算。类似地，从内存提取数据所用的时间以纳秒计算，而从磁盘提取数据的时间以毫秒为单位。

由于在性能上存在这样的数量级差异，所以，使用这些设备的最佳方式是：对于代码中性能至关重要的部分，要避免依赖这些设备。如使用本书中讨论的技术，开发人员编写的程序就能做到将访问辅助设备放在非关键路径上，或是对此作出合理调度，使系统计算部分在系统访问辅助设备的同时仍能处于活动状态，积极完成工作。

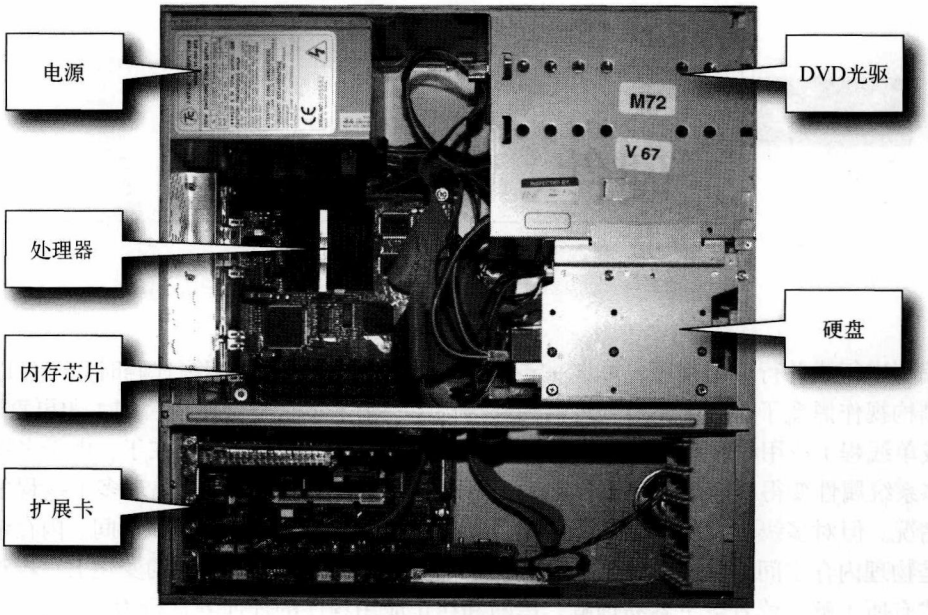

图1-1　个人计算机的内部结构

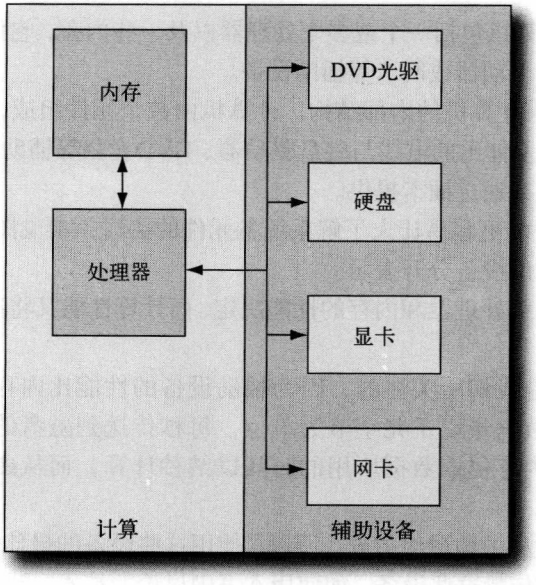

图1-2　个人计算机的抽象图示

1.2 多核处理器的缘起

微处理器已存在相当长时间了。x86架构可追溯至1978年发布的8086，而SPARC架构则出现得稍晚，第一个SPARC处理器于1987年面世。在那段时间，性能提升多源自加快处理器时钟速度（最初的8086处理器时钟频率约为5 MHz，而最新处理器的时钟频率约是其600倍，超过3 GHz）和架构方面的改进（同时发出多个指令等）。然而，最近的处理器却增加了CPU芯片上的*核心数*量，而非强调单个线程在处理器上运行时的性能提升。处理器核心是执行应用程序指令的部分，因此有了多个核心，单个处理器就能同时执行多个应用程序。

变更为多核处理器的原因很容易理解：提高串行性能日益困难。为使处理器更快地执行指令，要使用大量硅片面积，会增加耗电，产生更多热量。此外，尽管通过这种方法也有可能获得极大的性能提升，但通常性能提升有限，在10%至20%之间。相比之下，如不采用增加硅片面积，而是用这一面积来新添一个核心，则产生的处理器可能完成两倍的工作量，而拥有4个核心的处理器则能完成4倍的工作量。因此，提升整体性能最有效的方法是增加该处理器能支持的线程数量。显然，如何利用多个核心与其说是硬件问题，不如说是软件问题，不过如本书稍后所示，这是一个经过深入研究的软件问题。

多核处理器有关的术语颇为令人费解。大多数人熟悉的微处理器都是有许多引脚的黑色薄片。多处理器系统中有多个微处理器插至系统板上。当一个处理器只能运行单个线程时，处理器、CPU、芯片和核心四者的数量关系相对简单，即在同一系统中它们的数量相等，因而这些术语可以互换使用。但随着多核处理器的出现，情况有所变化。事实上，在多核处理器背景下可能很难对这些术语的确切定义达成共识。

本书将使用术语*处理器*（processor）和*芯片*（chip）来指代这个有许多引脚的黑色薄片。但用*插座*（socket）来称呼它的情况也并不少见。细心的读者可能已经注意到，这些都是可数的实体，你可以取下计算机机箱的盖子，数一下插座或处理器的数量。

单个多核处理器将为用户和操作系统提供多个*虚拟*CPU。虚拟CPU在物理上不可数，你不能打开计算机机箱查看一下主板，然后说出此主板能支持多少个虚拟CPU。但是，对于调度工作的操作系统来说，虚拟CPU却是可见的实体。

同样，我们也很难确定系统包含多少核心。如果拆开微处理器查看硅片，也许能确定核心数量，尤其是相关文档说明了应该有多少核心时！识别核心并非有规可循的学问。同样，我们也无法查看一下核心就确定此核心能支持多少个软件线程。由于单个核心可以支持多线程，核心的概念是否重要就值得商榷了，因为它对应的既不是有形的可数实体，也不是操作系统用来分配工作的虚拟实体。但实际上，要了解系统的性能，内核的概念很重要，随着学习你会逐渐明确这一点。

还可能让人迷惑的是*线程*这一术语。线程可指代硬件线程或软件线程。软件线程是指处理器执行的指令流，而硬件线程是指执行某个软件线程的硬件资源。多核处理器有多个硬件线程——这些都是虚拟CPU。其他资料可能称硬件线程为*缕程*（strand）。一个硬件线程能支持一个软件线程。

在某个系统上运行的软件线程通常比同时支持它们的硬件线程多得多，因此，这些线程中有许多处于不活动状态。当处于活动状态的软件线程比运行它们的硬件线程多时，操作系统将在活动的软件线程之间共享虚拟CPU。每个线程将运行很短的一段时间，然后操作系统将把它切换到做好运行准备的另一个线程。将线程转移到虚拟CPU或从虚拟CPU转移出来的行为称为*上下文切换*。

1.2.1　在单芯片上支持多线程

处理器的核心是芯片上负责执行指令的部分。核心有许多组成部分，我们将在本章稍后详细讨论其中某些部分。处理器的简化原理图见图1-3。

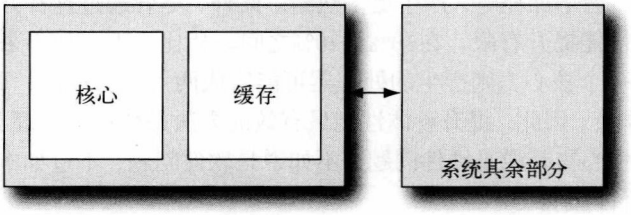

图1-3　单核处理器

缓存是芯片上保存最近使用数据和指令的部分。查看一下处理器内的硅片（如图1-7所示），可以看到核心和缓存是肉眼可见的两个元件。关于缓存，请参见1.2.3节。

让芯片运行多线程最简单的方式就是多次复制同一核心，如图1-4所示。最早能支持多线程的处理器就得益于这种方法。这也是多核处理器的基本思想。这是一种简单的方法，因为此法只需采用现有处理器的设计并加以复制，两个核心之间以及内核和系统之间的通信涉及的情况较为复杂，但核心（这是处理器最复杂的部分）的变化最小。两个核心共享一个与系统其余部分的接口，这意味着两个核心必须共享对系统的访问。

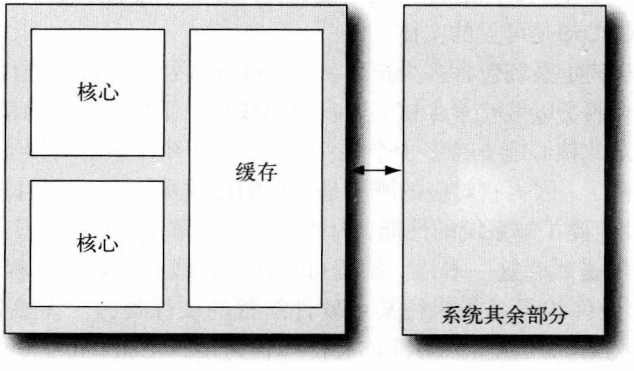

图1-4　能同时运行两个硬件线程的双核处理器

然而，这并非唯一的方法。另一种方法是使单核执行多个线程的指令，如图1-5所示。对这种设计存在多种改进：

- 该核心可以用100个时钟周期执行某个软件线程的指令，然后切换到另一个线程执行接下来的100个时钟周期；
- 该核心每个时钟周期可从两个线程之一提取指令，轮流进行；
- 该核心每个时钟周期可同时从多个线程各取一条指令；
- 该核心可在每次软件线程遇到长延迟事件时（如缓存未命中，此时必须从存储器中取数据）切换软件线程。

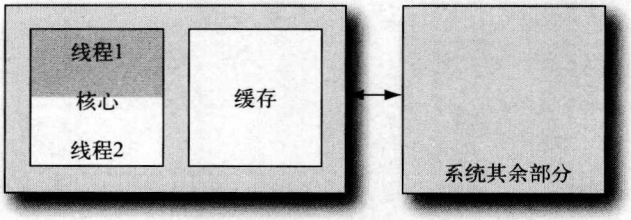

图1-5 有两个硬件线程的单核处理器

若两个线程共享一个核心，则每个线程将获得一份资源，份额大小取决于另一线程的活动和可用资源的数量。例如，如果一个线程停下等待内存，那么另一线程就可独占对所有核心资源的访问。然而，如果两个线程要同时发出相同类型的指令，那么对于某些处理器，只有一个线程能成功发出，另一个线程不得不等待下次重试。

多数多核处理器采用多个核心与单核心多线程相结合的方式，最简单的例子就是带两个核心、每个核心支持双线程的处理器，这样整个处理器就能支持4个线程。图1-6显示了此种配置。

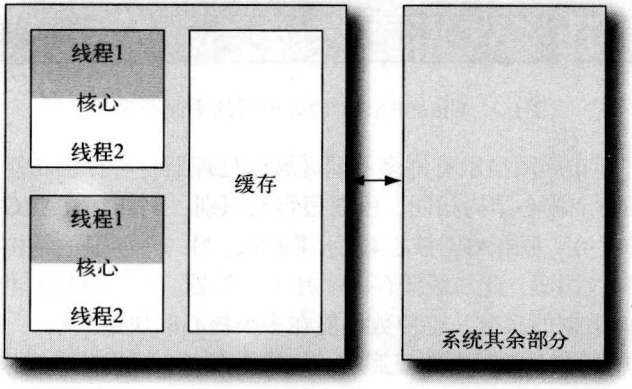

图1-6 共有4个硬件线程的双核处理器

处理多线程的能力反映在操作系统中，通常表现为该系统具有许多虚拟CPU。因此，从用户