微软技术丛书

Microsoft

Visual Studio 2010 并行编程从入门到精通

（美）Donis Marshall 著
梁春艳 译

Step by Step

清华大学出版社

微软技术丛书

Microsoft

Visual Studio 2010 并行编程从入门到精通

（美）Donis Marshall 著
梁春艳 译

Step by Step

清华大学出版社
北京

内 容 简 介

多核架构是当前的行业趋势，越来越多的读者希望自己写的程序能够在这样的架构上取得良好的性能。本书针对并行编程，沿袭深受读者欢迎的 STEP-BY-STEP 风格，一次讲授一个知识点，由浅入深地介绍了相关基础知识，如任务并行和数据并行，讨论了并发集合和线程同步，阐述了如何使用 Visual Studio 2012 来维护和调试并行应用。

本书可帮助.NET 开发人员理解并行编程及相关技术的核心概念，帮助他们开发高性能的并行应用。

图书在版编目(CIP)数据

Visual Studio 2010 并行编程从入门到精通/(美)马歇尔(Marshall，D.)著；梁春艳译. --北京：清华大学出版社，2013
(微软技术丛书)
书名原文：Parallel Programming with Microsoft Visual Studio 2010 Step by Step
ISBN 978-7-302-30522-4

Ⅰ. ①V… Ⅱ. ①马… ②梁… Ⅲ. ①程序语言—程序设计 Ⅳ. ①TP312

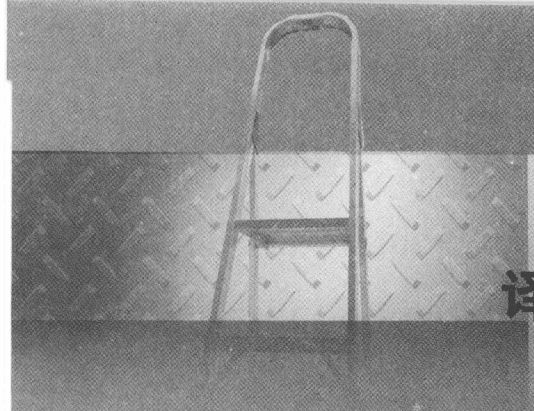
中国版本图书馆 CIP 数据核字(2012)第 258064 号

责任编辑：文开琪
封面设计：杨玉兰
责任校对：周剑云
责任印制：沈 露

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-62770175 邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn,010-62791865

印 装 者：北京密云胶印厂
经 销：全国新华书店
开 本：185mm×260mm 印 张：12.75 字 数：265 千字
版 次：2013 年 1 月第 1 版 印 次：2013 年 1 月第1 次印刷
印 数：1～4000
定 价：39.00 元

产品编号：040289-01



译 者 序

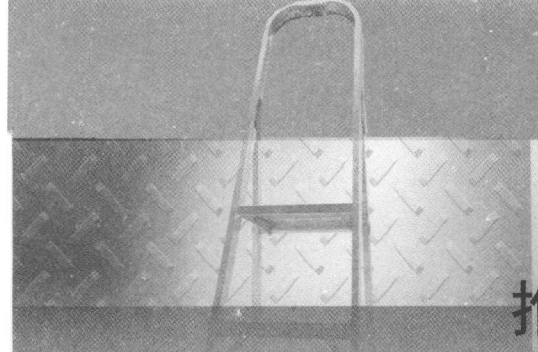
本书介绍了如何使用 Visual Studio 2010 来创建并行应用程序，详细讨论了 TPL 和并行编程概念，基本涵盖了并行编程领域的基础知识。与此同时，本书还讨论了并发集合和线程同步，指导读者如何使用 Visual Studio 来维护和调试并行应用程序。除了说明性的内容之外，大部分章节还包括 STEP-BY-STEP 实例以及可从网上免费下载的示例项目。通过学习每一个例子，尽可能帮助读者学习并行编程的概念和技术。本书还提供书中所有例子的源代码，读者可从网上免费下载。

在本书的翻译过程中，对于英文版中明显存在的少数纰漏（主要是排印或疏漏），都一一进行了修正。这些错误一般都很明显，因此译文中没有特别声明。

为了保证译文的准确性，使译文更加通俗易懂，我在翻译过程中勤查资料，也花相当多的时间来推敲和斟酌文字。但毕竟翻译水平有限，书中也许还存在一些疏漏，在此恳请广大读者批评指正。任何问题都可发送到 hhlcy328@gmail.com。

这里，我首先要感谢本书作者 Donis Marshall 写出了这样一本非常出色的并行编程教材。还要感谢翻译过程中涉及的所有人士，他们是梁法明、荆秀梅、文天敏、张时全。感谢我的父母对我工作的支持。同时还要感谢我的先生胡辉，他在我的翻译工作中提供了很多帮助。最后感谢我的女儿胡清扬，她天真可爱，聪明伶俐，给我的家庭生活带来了无穷的欢乐。

最后，衷心祝愿读者朋友能通过本书开始愉快而激动人心的并行编程之旅！



推 荐 序

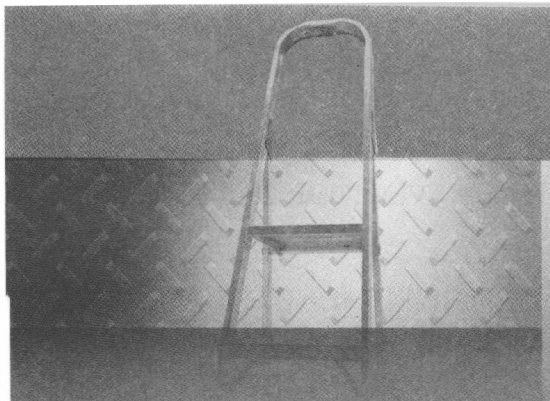
Tracy Monteith

本书从平淡的硬件和管线说起，接着讲到软件赋予硬件的令人兴奋的、美妙的魔力。我的第一个软件程序写好后，就排队等着学校那台唯一的计算机完成后面的一系列任务。同年，PC 横空出世，为企业和家庭(以前是学术界、银行和政府)提供了负担得起的计算能力。一个全新的世界以及后来的职业生涯，都向我表明一点：美妙的代码，用不着等待。程序一写完，我就可以庆祝结果。所以，我又写了一个程序，一个，又一个。

幼年时期，我们就学会了用线性方式来解决数学问题，所以“先做这个，然后再做那个”这样的顺序概念几乎是全球程序员的时代精神。但是，时代变了，由于计算机再也不像我们人脑一样抱有计算偏见，而是将线性编程和顺序编程联系起来，得出一个需要充分利用并行处理优势的设计方案，这样的设计方案需要我们思考新的方法。为了产出快速、安全、可靠的全球化软件，程序员需要新的工具作为当前编程方法的补充。如此这般，本书便应运而生。

Donis Marshall 运用叙事方法将他的专长娓娓道来，既有并行编程基础知识，又有行之有效的决策条件(用于充分利用并行编程的威力)。Donis 写过 6 本编程方面的书，有丰富的行业实践经验，开设过十几门编程课程，因此，他为刚涉足并行编程的新手开发人员提供的是最好的入门参考。STEP-BY-STEP 的体例，Donis 的叙事性写作风格，能够很好地帮助读者获得技能和经验的提升。

并行编程的世界正迅速波及到热切盼望(以任何可能的方式)充分利用现代计算机体系结构的每一位开发人员。站在巨人的肩上，.NET Framework 4 延续其为开发人员和系统工程师系统提供新功能的传统。这些新工具提供了强大的功能，同时也是一个很大的挑战，即如何以及在哪里充分利用好它们。本书可以确保全球的程序员能够有效地将并行编程加入他们的设计组合之中。



前　言

并行编程的确重新定义了多核架构的编程模型，这已属老生常谈。出于这个原因，并行编程已经被提升为.NET Framework 4 的核心技术。在.NET Framework 4 中，任务并行库(TPL)和 *System.Threading.Tasks* 命名空间包含了并行编程的实现。与此同时，Visual Studio 2010 已经得到了增强，现在包含几个帮助创建和维护并行应用程序的功能。如果你是一个微软的开发人员，想把应用程序分解为在不同处理器内核上运行的并行任务，那么 Visual Studio 2010 和 TPL 正是你需要的工具。

本书完整介绍了如何使用 Visual Studio 2010 来创建并行应用程序，详细讨论了 TPL 和并行编程概念，但本书仍然属于入门读物——它涵盖的是并行编程各领域的基础知识，如任务并行和数据并行。虽然这本书没有全面覆盖每一个并行编程主题，但它的确在如何使用并行编程概念方面提供了必要的指导。

除了涵盖核心并行编程概念，本书还讨论了并发集合和线程同步，指导你使用 Visual Studio 来维护和调试并行应用程序。除了说明性的内容之外，大部分章节包括动手练习和可下载的练习文件，供读者进一步探索。

本书为谁而写？

本书的目的是帮助 Visual Basic 和 Visual C#开发人员理解并行编程以及相关技术的核心概念。对那些想利用多核架构的程序员，它尤其有用，更何况多核架构是当前的行业趋势。本书要求读者略懂.NET Framework，但可以没有任何并行编程经验。本书对那些已经熟悉并行编程概念并想知道 TPL 最新功能的人也很有用。

本书不适合谁？

不是每本书都适合全民阅读。作者必须首先了解读者的知识水平，这样才能避免自己写出来的东西让高端读者厌烦或者失去入门级读者。

假设

本书希望你至少对.NET 开发和面向对象编程概念略知一二。TPL(即使不是对全部)对大多数.NET Framework 4 平台可用。本书只包含 C#范例，不过，这些例子只需很小的改动就可移植到 Visual Basic .NET 上。如果你还没有学习这些语言，可以考虑阅读入门级畅销

书《Visual C# 2010 从入门到精通》(John Sharp 著，周靖译)或者《Visual Basic 2010 从入门到精通》(Michael Halvorson 著，汤涌涛译)。

虽然本书主要关注并发编程概念，但它也假设你对线程和线程同步概念有最基本的了解。如果想在本书之外或者想扩展线程方面的知识，可以考虑阅读《CLR via C#》(Jeffrey Richter 著，周靖译)。

本书的组织

本书共 7 章，每章关注不同的主题或者并行编程的相关技术。

- 第 1 章“并行编程介绍”介绍并行编程的基本概念。
- 第 2 章 “任务并行”的重点是创建并行迭代并且将顺序循环重构为并行任务。
- 第 3 章 “数据并行”的重点是从不同的操作创建并行任务。
- 第 4 章 “PLINQ”是一个使用语言集成查询(LINQ)的并行编程概述。
- 第 5 章 “并发集合”解释如何使用并发集合，比如，ConcurrentBag 和 ConcurrentQueue。
- 第 6 章 “自定义”演示自定义 TPL 的技术。
- 第 7 章“报告和调试”展示如何调试和维护并行应用程序并且充分讨论并行编程。

找到阅读本书的最佳起点

本书涵盖.NET Framework 并行编程相关的广泛的技术和概念。根据你的需求以及对.NET Framework 4 并行编程的熟悉程度，你可能希望专注于本书的特定领域。可以使用下表确定以怎样的方式来学习本书。

读者需求	阅读建议
了解并行编程的概念	从第 2 章开始并阅读本书的剩余部分
熟悉.NET Framework 3.5 的并行扩展	• 如果你需要复习核心概念，阅读第 1 章 • 略过第 2 章和第 3 章任务和数据并行的基本概念 • 阅读第 3 章到第 7 章，探究 TPL 的详细内容
对 LINQ 数据提供者有兴趣	阅读第 4 章以及第 7 章
对自定义 TPL 有兴趣	阅读第 6 章

本书大部分章节包含可实际操作的示例，供你动手尝试刚刚学习的概念。无论选择关注哪些章节，务必确保事先已经在系统上下载并安装了示例应用程序。

本书约定和特点

本书采用以下约定使文字更容易阅读和理解。

- 每个练习都由一组任务构成，采用编号步骤(1，2 等)来展示。必须按步骤操作来完成整个演练。
- 大部分练习的结果都在一个控制台窗口显示，供读者进行比较。
- 每个练习的完整代码都在练习的结尾处出现。大部分代码都可以从网上下载
- 每章都还用一个“快速参考”小节做总结，回顾这一章的重要细节，简要概述这一章的内容。
- 由于译者硬件和服务环境有限，部分截图只能沿用英文截图，还请读者见谅。

系 统 要 求

需要准备好以下硬件和软件环境来完成本书的实战练习。

- 下列中的一个：带有 Service Pack 3 的 Windows XP 、带有 Service Pack 2 的 Windows Server 2003、带有 Service Pack 1 或者更高版本的 Windows Vista、Windows Server 2008、Windows Server 2008 R2、Windows 7 或者 Windows 7 SP1。
- Visual Studio 2010 的任何版本(如果使用的是速成版产品，可能需要多个下载)。
- 1.6-GHz 计算机或者更快的处理器(推荐 2 GHz)。
- 1 GB (32-bit) 或者 2 GB (64-bit) RAM(如果在虚拟机上运行，增加 1 GB)。
- 3.5 GB 可用硬盘空间。
- 5400-RPM 硬盘驱动器。
- DVD-ROM 驱动器(如果从 DVD 安装 Visual Studio)。
- Internet 连接，用于下载练习所需要的范例代码。

根据具体的 Windows 配置，可能需要有本地管理员权限来安装或配置 Visual Studio 2010。

代 码 示 例

本书大多数实战演练都允许交互式尝试在正文中学到的新内容。所有示例项目(演练前后有两个版本)都可从以下网址下载：

http://go.microsoft.com/FWLink/?Linkid=222678

单击页面中的 *Download the companion content* 链接，然后下载.zip 文件。

注意 除了代码示例，你的系统应该已经安装了 Visual Studio 2010。

安装代码示例

按照这些步骤在你的计算机上安装代码示例，以便可以使用它们完成本书中的练习。

1. 解压从本书网站下载的 Parallel_Programming_Sample_Code.zip 文件。

2. 如果出现提示，审查显示的终端用户许可协议。如果你接受这些条款，选择“接受”选项，然后点击“下一步”按钮。

注意 可以从下载 Parallel_Programming_Sample_Code.zip 文件的网页访问它。

致　　谢

我想感谢以下人员： Russel Jones，为他的无限耐心；Ben Ryan，为他的另一个为微软出版社编写的极好机会；Devon Musgrave，为他的最初的指导。还有很多支持我的朋友：Paul、Lynn、Cynthia、Cindy 以及其他，Adam、Kristen 和 Jason，他们是宇宙中最明亮的星星。

勘误表和图书支持

我们已经做了很多努力来确保本书及伴随内容的准确性。自本书出版后报告的任何错误都列在微软出版社位于 oreilly.com 的网址：

http://go.microsoft.com/FWLink/?Linkid=223769

如果发现任何还没有列出来的错误，可以通过相同的页面报告给我们。如果需要额外的帮助，可以给微软出版社图书支持 *mspinput@microsoft.com* 发送电子邮件。

请注意，前面的地址不提供微软软件的产品支持。

我们期待您的来信

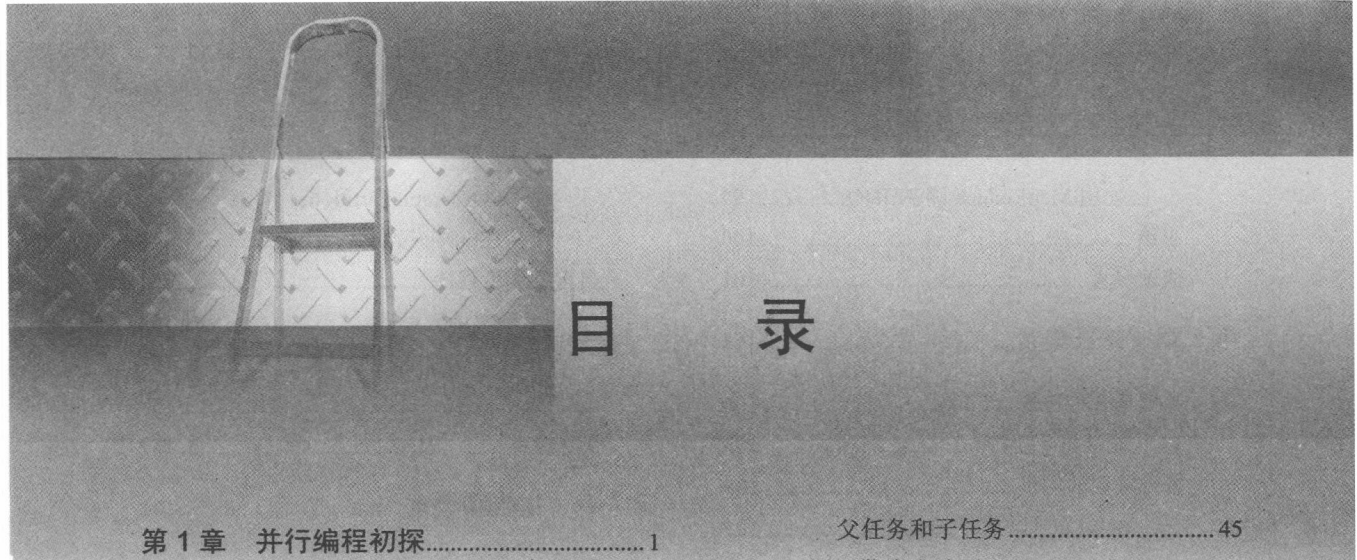
在微软出版社，读者的满意是我们的首要任务，读者的反馈是我们最宝贵的资产。请告诉我们您对本书的看法：

http://www.microsoft.com/learning/booksurvey

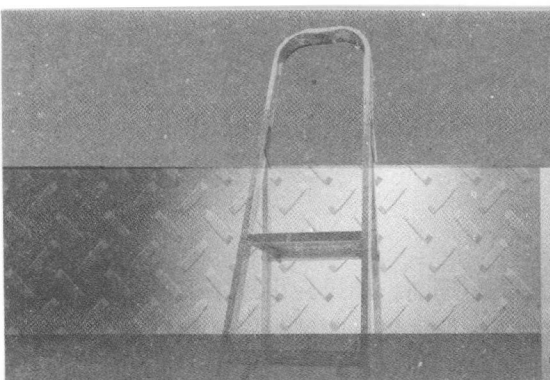
这项调查很短，我们会阅读您的每一个意见和想法。感谢您的反馈意见！

保 持 联 系

让我们继续交流！我们的 Twitter 专页：*http://twitter.com/MicrosoftPress*



目　录



第 1 章
并行编程初探

学习目标

- 解释并行编程的目标，各种硬件架构以及并发和并行编程的基本概念
- 定义并行和性能之间的关系
- 使用阿姆德尔定律计算加速
- 使用古斯塔夫森定律计算加速
- 识别和应用并行开发设计模式

并行编程将改变 PC 的计算天地。这是一个颇有气势的声明。但不管怎样，它体现的是将并行计算由学术界、科学实验室和更大的系统移向桌面所带来的影响。并行编程的目标是通过优化可用处理器内核的使用(即内核的并行执行)来提高性能。随着处理器速度不断提升的态势逐渐减缓，这个目标变得越来越重要。

摩尔定律预言每平方英寸集成电路的晶体管容量每两年就会提高一倍。戈登·摩尔在 20 世纪 60 年代中期宣布和预测这一趋势将持续至少 10 年，但是摩尔定律实际上已经适用了将近 50 年。摩尔定律的预测常常被解释为处理器速度每隔几年会翻一番。然而，摩尔定律的地基开始出现裂缝。开发人员现在必须找到满足客户需求的其他手段以实现更快的应用、附加功能以及更大的范围。并行编程是一种解决方案。这样，摩尔定律将在无限的未来继续有效。

微软认识到了并行编程在未来的重要作用。这就是为什么并行编程由一个扩展提升为公共语言运行时(Common Language Runtime，CLR)的核心组成部分的原因。新功能已经被添加到.NET Framework 4 和微软 Visual Studio 2010 以支持并行编程。这是在承认一个事实，即随着 PC 中可用多核处理器的增加，并行编程正在迅速变成一种主流技术。

并行代码相对于相同应用的顺序版本或者新的应用程序，开发无疑更加复杂。新的调试窗口特别添加到 Visual Studio 2010 以帮助维护和调试并行应用程序。“并行任务”和“并行堆栈”窗口有助于从并行执行和任务的上下文解释应用程序。为优化性能，Visual Studio 分析器和并发可视化工具一起分析并行应用程序，并且提供图形和报告来帮助开发人员隔离潜在的问题。

并行编程是一个广泛的技术领域。一些软件工程师在自己的职业生涯中一直在研究和实现并行代码。幸运的是，.NET Framework 4 提取了大部分细节，使得开发人员可以专注于为企业和个人需求编写并行应用程序而提取大部分内部细节。尽管这样，理解并行编程的目标、约束和潜在动机是非常有帮助的。

多核计算

过去，软件开发人员受益于单核计算机新硬件性能的不断提高。如果应用程序很慢，仅仅等待就可以了——因为硬件性能的提高，它不久就会运行得更快。应用程序只是简单地依赖更好的性能。然而，我们不能再依赖硬件的不断提高来确保应用程序取得更高的性能了！

新一代处理器内核的性能改善已经放慢脚步，我们现在受益于多核架构的可用性。这使得开发人员能够继续实现性能的提高并且继续提升应用程序的速度。不管怎样，确实需要编程范式有所改变，这正是编写本书的目的。

目前，双核和四核机器是事实上的标准。今天，在北美以及其他地区，你可能无法(并且也不会想)在当地的计算机商店购买单核台式机。

单核计算机的一些制约因素阻止了性能增益的继续，而这在过去是可能的。主要制约因素是处理器速度和散热量的相互关系。随着处理器速度的增加，热量会不成比例地增加。这就对处理器的速度设置了一个实际的门槛。显著提高计算能力而无散热问题的解决方案还没有找到。多核架构是一种替代方法，多个处理器核共享一个芯片模。额外的核提供了更强大的计算能力而不会有散热问题。在并行应用程序中，可以利用多核架构获得潜在的性能增益而不会有相应的散热问题。

多核 PC 已经改变了计算行业。直到最近，单核计算机一直是 PC 最流行的架构。但是情况正在迅速发生改变，除了是 PC 计算体系结构的下一个进化步骤外并不代表什么。多核架构和并行编程的结合会将摩尔定律继续存在于可预见的未来。

随着诸如英特尔超线程技术等技术的出现，每个物理内核变成两个或者潜在的更多的虚拟内核。例如，一个具有四个物理内核的机器似乎具有八个逻辑内核。物理内核和逻辑内核之间的区别对开发人员和用户来说是透明的。在未来的 10 年中，我们可以预期，在标准的 PC 中，物理处理器内核和虚拟处理器内核的数量将有显著的增长。

多指令流/多数据流

1966 年，迈克尔·弗林提出了一个分类法，描述各种硬件架构中并发指令和数据流之间的关系。该分类法被称为“弗林分类法”，有如下一些分类。

- **SISD (单指令流/单数据流)** 该模型具有单指令流和单数据流，使用单核处理器描述计算机架构。
- **SIMD (单指令流/多数据流)** 该模型具有单指令流和多数据流。模型将该指令流应用到每一个数据流。相同指令流的实例能够在多个处理器内核中并行运行，服务不同的数据流。例如，当对多个输入值应用相同算法的时候，SIMD 很有用。
- **MISD (多指令流/单数据流)** 该模型具有多指令流和单数据流，且对单个数据源应用多个并行操作。例如，该模型可以被用来对单个数据源运行各种解密程序。

- **MIMD (多指令流/多数据流)**　该模型具有多指令流和多数据流。在多核计算机中，每个指令流与独立的数据运行在单独的处理器中。这就是多核 PC 的当前模型。

MIMD 模型可以进一步细化为多程序/多数据(MPMD)或单程序/多数据(SPMD)。在 MPMD 子类中，每个处理器独立执行不同的进程。在 SPMD 中，进程被分解为单独的任务，每个任务代表程序中不同的位置。任务在单独的处理器内核中执行。这是当今多核 PC 的主流架构。

下表给出了弗林分类法。

弗林分类法	单数据流	多数据流
单指令流	SISD	SIMD
多指令流	MISD	MIMD

有关弗林分类法的额外信息可以从它的维基页面获得：

http://en.wikipedia.org/wiki/Flynn%27s_taxonomy

多线程

线程代表程序中的行为。进程本身什么也不做；相反，它承载正在运行应用程序所消耗的资源，比如堆和栈。线程是应用程序中一种可能的执行路径。线程能够执行独立的任务或者配合相关任务的操作。

并行应用程序也是并发的。然而，并不是所有并发的应用程序都是并行的。

并发应用程序能够在单内核上运行，而并行执行需要多个内核。这个区别背后的原因叫“交织”(interleaving)。当多个线程并发运行在单处理器计算机上时，Windows 操作系统基于线程的优先级以及其他因素，以一种轮转(round-robin)的方式交织执行线程。在这种方式下，该处理器在几个线程间共享。我们可以认为它是逻辑并行的(logical parallelism)。在物理并行情况下，有多个内核，工作被分解为任务并且在不同的处理器内核中并行执行。

当线程被另一个线程中断时，线程被抢占。此时，正在运行的线程让步于下一个线程。在这种方式下，线程在单处理器中交织。当一个线程被抢占，操作系统保留正在运行线程的状态并且载入下一个即将执行线程的状态。在处理器中，交换运行的线程引发了上下文切换(context switch)以及内核和用户模式之间的过渡。上下文切换的开销是比较大的，因此减少上下文切换的数量对于提高性能很重要。

线程被抢占有下列几个原因：

- 一个更高优先级的线程需要运行
- 执行时间超过限量

- 收到输入输出请求
- 线程自发让出处理器
- 线程被一个同步对象阻塞

即使是在单处理器的机器上，并发执行也有下述优点：

- 多任务
- 响应迅速的用户界面
- 异步输入/输出
- 改进的图形渲染

并行执行需要多个内核以便线程可以并行执行而无需交织。理想情况下，每个可用的处理器都需要一个线程。然而，这并不总是可能的。当线程的数量超过可用处理器的数量时，超额认购(oversubscription)的情况会发生。当发生这种情况时，共享同一个处理器的线程就会发生交织执行的情况。

相反地，当线程数量少于可用处理器数量时，不足认购(undersubscription)就会发生。当这种情况发生时，就会有空闲处理器和低于最佳的 CPU 使用率。当然，我们的目标是最大化 CPU 使用率，同时平衡超额认购或不足认购所引起的潜在的性能下降。

如前所述，上下文切换会对性能造成负面影响。然而，一些上下文切换相对其他切换而言更加昂贵；其中一个更昂贵的上下文切换是跨核上下文切换。一个线程可以运行在一个专用处理器上，也可以跨处理器。由单个处理器服务的线程都有处理器关联(processor affinity)，这样会更加有效。在另一个处理器内核抢占和调度线程会引起缓存丢失，作为缓存丢失和过度上下文切换的结果要访问本地内存。总之，这称为“跨核上下文切换”(cross-core context switch)。

同步

多线程涉及的不只是新建多个线程。启动一个线程所需要的步骤比较简单。但为一个线程安全的应用程序管理这些线程是一个更大的挑战。同步是用来新建线程安全环境的一个最常见的工具。

即使是单线程的应用程序，有时也会使用同步。例如，单线程应用程序可能会同步内核模式的资源(它们是跨进程共享的)。然而，同步在多线程应用程序中更为常见，在这类应用中，内核模式和用户模式的资源也许都会被争用。共享数据是多个线程间争用的第二个原因，而且也有同步的需求。

大多数同步由同步对象完成。有专用同步对象，如互斥、信号量和事件。同样可用于同步的通用对象包括进程、线程和注册表项。例如，无论线程是否已经执行完毕都可以同步。大多数同步对象是内核对象，它们的使用需要上下文切换。诸如临界区这样的轻量级同步对象是用户模式的对象，可以避免昂贵的上下文切换。在.NET Framework 中，*lock* 语句和 *Monitor* 类型是本地临界区的封装。

当一个线程在一段时间内都无法获得同步对象或者访问共享数据时，就会出现争用(contention)情况。该线程通常会阻塞直到实体可用。当争用情况很短时，相关的同步开销是比较昂贵的。如果短期争用很多，这样的开销就不能忽略。这种情况下，可以使用自旋(spinning)来代替阻塞。应用程序可以选择在用户模式下自旋，消耗 CPU 周期但避免内核模式切换。片刻后，该线程可以重新尝试获取共享资源。如果争用很短，我们可以在第二次尝试中成功获得资源从而避免阻塞以及相关的上下文切换。同步自旋被认为是一种轻量级的同步，微软为此已经在.NET Framework 中添加了 *SpinWait* 结构这样的类型。例如，自旋结构用于 *System.Collections.Concurrent* 命名空间的很多并发集合中，用来新建线程安全和无锁的集合。

大多数并行应用程序依赖某种程度的同步。开发人员经常将同步视为“必要之恶”(necessary evil)。同步的过度使用是很遗憾的，因为大多数并行程序在不妨碍并行运行时执行得最好。通过同步来序列化并行应用程序与总体目标背道而驰。事实上，并行应用程序速度提高的潜力受限于按顺序运行的应用程序所占的比例。例如，当 40%的应用程序按顺序执行时，理论上，最大的可能是速度提高 60%。大多数并行应用程序以最小的同步启动。不管怎样，同步往往是任何问题的首选解决方案。这样一来，同步就像树上的苔藓一样迅速传播开来。在极端的情况下，其结果是一个复杂的顺序应用程序(由于某种原因而有多个线程)。在你自己的程序中，要尽量让并行应用程序能够并行。

加　　速

加速(speedup)是在多核机器上运行应用程序(相比单核机器)所能够预期的性能收益。测量加速时，单核机器的性能为基线。例如，假设一个应用程序在单核机器上的持续时间是六小时，那么该应用程序在一个四核机器上运行时，持续时间减少到三个小时。加速等于 2—(6/3)—，换句话说，应用程序快了两倍。

你可能会认为在单核机器上运行的应用程序在双核机器上运行时会快两倍，在四核机器上运行时会快四倍。但是，这并不完全正确。除了一些特例，比如超线性加速，即使整个应用程序都并行运行，线性加速也是不可能的。这是因为总有一些并行应用程序的开销，比如将线程调度到单独的处理器。因此，线性加速是不可实现的。

并行代码线性加速具有以下限制：

- 串行代码
- 并行开销
- 同步
- 顺序输入/输出

预测加速在设计、基准评测以及测试并行应用程序时很重要。幸运的是，有计算加速的公式。其中一个公式是阿姆德尔定律。吉恩·阿姆德尔在 1967 年新建了阿姆德尔定律，用以计算并行应用程序的最大加速。

阿姆德尔定律

阿姆德尔定律基于三个变量来计算并行代码的加速：

- 在单核机器上运行应用程序的持续时间
- 并行应用程序的百分比
- 处理器内核的数量

下面是公式，它返回单核对多核性能的比率：

$$\text{Speedup} = \frac{1}{1-P+(P/N)}$$

该公式以应用程序在单核机器上的持续时间为基准。

公式中的分子代表基础持续时间，总是等于 1。计算的动态部分在分母。变量 P 是并行应用程序的百分比，N 是处理器核的数量。

举一个例子，假设有一个应用程序，75%并行且运行在一个具有三个处理器内核的机器上。计算阿姆德尔定律的第一次迭代如下所示。在公式中，P 是.75(并行部分)，N 是 3(内核数量)。

$$\text{Speedup} = \frac{1}{(1-.75)+(.75/3)}$$

可以将其简化为：

$$\text{Speedup} = \frac{1}{.25+.25}$$

最终结果是，加速等于 2。该应用程序在 3 个处理器内核的机器上运行时会快两倍，即 Speedup = 2。

加速以可视化来展示有助于解释阿姆德尔定律的含义。在下图中，加速的计算用图形表示。持续时间表示为相同长度的单元。在单核机器上，应用程序的持续时间是四个单元。其中一个单元包含的代码必须顺序执行。

这意味着应用程序的 75%可以并行运行。此外，在这种情况下，有 3 个可用的处理器内核。因此，这 3 个并行单元可以并行运行并且合并为 1 个单元的持续时间。所以，应用程序的顺序和并行部分都需要一个单元的持续时间。因此，总共需要 2 个单元的持续时间——比原来的 4 个单元减少了——加速为 2。这样，应用程序的运行就快了两倍。这就证实了前面使用阿姆德尔定律的计算。