CUDA 是 C 语言的扩展，允许 GPU 代码以常规 C 语言编写。代码是针对中央处理器（CPU）或针对图形处理器（GPU）的。该主机处理器在 GPU 设备上产生多线程任务（或 CUDA 中已知的内核）。GPU 具有自己的内部调度器，然后将内核分配给当下的任何 GPU 硬件。 稍后将详细介绍调度。 如果在任务中有足够的并行性，程序的速度也应该如 GPU 中的 SM（流处理器）的数量增长。

但是，这里隐藏着一个大问题。 你必须了解程序中可以并发运行的代码的百分比。 最大的加速可能受到串行代码数量的限制。 如果你有无限计算处理能力，可以在瞬间完成并行任务，您仍然要留下一些时间来运行串行部分的代码。 因此，如果我们确实可以并行化，我们必须首先考虑大量的计算量。

NVIDIA 致力于为 CUDA 提供支持。值得参考的信息、例子和帮助开发的工具可从网站 http://www.nvidia.com 的 CudaZone 模块中下载。CUDA 与其前身不同，现在实际上已经开始获得势头，这是第一次看起来会有一种编程语言将成为 GPU 编程的首选。鉴于支持 CUDA 的 GPU 的数量现在达到数百万，所以有一个巨大的市场在那里等待支持 CUDA 的应用程序。

目前有许多支持 CUDA 功能的应用程序，并且这个数量日益增长。NVIDIA 在社区网站 http://www.nvidia.com/object/cuda_apps_flash_new.html 上展示了许多这些应用程序。在程序必须做很多计算工作的领域，例如从你的家庭电影制作 DVD（视频转码），我们现在看到的许多主流视频包都支持 CUDA。CUDA 在这些领域的平均加速时间是 5 到 10 倍。

CUDA 的替代品：

OpenCL

那么其他的 GPU 制造商呢，ATI（现在的 AMD）就是最好的例子。 AMD 的产品范围与 NVIDIA 的原始计算机功率一样令人印象深刻。不过 AMD 推出了它的流计算技术到市场上，是在 NVIDIA 推出 CUDA 很长一段时间的之后。结果，相较于 AMD / ATI 流计算技术，NVIDA 有更多的可用于 CUDA 的应用程序。

OpenCL 和 Direct 计算不是我们在本书中所讨论的，但是值得一提在 CUDA 的替代品方面。CUDA 目前仅在 NVIDIA 硬件上正式可执行。虽然 NVIDIA 在 GPU 市场中占有相当大的一部分，但其竞争对手也拥有相当大的一大部分。作为开发商，我们希望开发尽可能大的市场的产品，特别是当我们在谈论消费市场的话题。 因此，人们应该意识到有 CUDA 的替代品会同时支持 NVIDIA 和其他硬件。

OpenCL 是 NVIDIA，AMD 和其他公司支持和开放的免版税标准。该 OpenCL 商标由苹果拥有。 它规定了允许使用计算的开放标准设备。 计算设备可以是 OpenGL 驱动程序的 GPU，CPU 或其他专用设备。 截至 2012 年，OpenCL 支持了所有主要品牌的 GPU 设备，包括了至少有 CPUSSE3 支持的 GPU。

熟悉 CUDA 的人可以比较容易地掌握 OpenCL，它们在基础概念上非常相似。然而，OpenCL 在使用上比 CUDA 要复杂得多，程序员需要在 OpenCL 中显式执行 CUDAruntime API 所做的工作。

你可以在 http://www.khronos.org/opencl/上阅读更多关于 OpenCL 的信息。现在还有大量有关 OpenCL 的书籍。在 CUDA 之前，我个人推荐在 OpenCL 之前学习 CUDA，因为 CUDA 对于 OpenCL 来说是更高级别的语言扩展。

DirectCompute

DirectCompute 是微软替代 CUDA 和 OpenCL 的平台。 它是一个与 Windows 操作系统关联的专有的产品，特别是 DirectX 11 API。对于任何之前编写过视频卡的人来说，DirectX API 是一个巨大的飞跃。 这意味着开发人员必须只学习一个 API 库来编写所有显卡，而不是给每个主要视频卡制造商的驱动程序编写程序或发布许可。

DirectX 11 是最新的标准，并支持 Windows 7。在这个标准里有 Microsoft 的名头，你可以在开发人员社区中看到一些相当迅速的意见采用，特别是这个开发者已经熟悉 DirectX API 的情况。如果你熟悉 CUDA 和 DirectCompute，那么将 CUDA 应用程序移植到 DirectCompute 是很简单的。根据微软的说法，如果你这样做，你通常可以在一个下午通过一些功课来熟悉两个系统。 然而，以 Windows 为中心，我们将从 UNIX 主导的许多高端系统排除 DirectCompute。微软还将推出一套额外的标准模板库（STL）Ctt AMP，这可能吸引更多熟悉 Ctt 风格 STL 的程序员。

CPU（解决方案）的选择

主要的并行处理语言扩展是 MPI 和 OpenMP，如果你是 Linux 开发者则会用到 pthreads 对于 Windows，有 Windows 线程模型和 OpenMP。 MPI 和 pthreads 在 Unix 可为各种端口提供支持。

MPI（消息传递接口）可能是最广为人知的消息接口。 它是基于过程的，并且在大型计算实验室中普遍应用。 它需要一个管理员正确配置安装，最适合受控环境。 并行度是通过在一组节点上产生数百个进程并明确的进行消息交换，通常通过高速的基于网络的通信链路（以太网或无限带宽技术）。 MPI 被广泛使用和传授。 这是一个集群受控环境中的很好的解决方案。

OpenMP（开放多线程）是一种设计用于在一个节点内并行的系统电脑系统。 它的工作方式完全不同，程序员指定了各种各样的并行指令通过编译器编译指示。然后，编译器会自动尝试根据可用的处理器核心数量将问题分解成 N 个部分。许多编译器都支持 OpenMP，包括用于 CUDA 的 NVCC 编译器。OpenMP 的由于底层 CPU 体系结构，容易造成缩放问题。因为 CPU 中的带宽不足以让所有的内核不断地将数据流传输。

Pthreads 是一个在 Linux 上用于多线程应用程序的库。和 OpenMP 一样，pthreads 使用线程而不是进程，因为它被设计用于并行化单个节点。然而与 OpenMP 不同，程序员负责线程管理和同步。这提供了更多的灵活性，因此能以更好的性能运行精心编写的程序。

ZeroMQ（0MQ）也值得一提。这是一个你可以链接到的简单库，我们将在本书的后面使用它来开发一个多节点，多 GPU 的例子。ZeroMQ 支持使用基于线程的单个交叉平台，过程基于网络的通信模型 API。它也可在 Linux 和 Windows 平台上使用。它是为分布式计算设计的，因此连接是动态的，会出现节点失败。

Hadoop 也是你可以考虑的。Hadoop 是 Google 的开源版本 MapReduce 框架。它主要针对 Linux 平台。这个框架的概念是把采集到的一个巨大的数据集，将其分成（或映射）成多个块。数据集使用并行划分成数百个或数千个节点文件系统，而不是单纯的发送数据到节点。取而代之的是将包含数据的程序发送到节点。输出将写入本地节点并保留在该节点。随后的 MapReduce 程序以前的输出，并以某种方式再次转换。由于数据实际上是多个节点的镜像，这允许高度容错以及高吞吐量系统。