第 3 章　并行计算机

3.1　概　　述

算法和多处理器架构相互之间是紧密联系的。在考虑并行算法的时候不能脱离将要支持这个算法的并行硬件。反过来，在考虑并行硬件的时候也不能脱离将要运行于其上的并行算法。计算机系统中可以通过硬件和软件的手段在不同的层次实现并行：

(1) 数据级并行，在这一层我们对一个数据的多个位或多个数据同时进行操作。例如位并行加法，乘法，以及二进制数的除法，向量处理器和处理多数据单元的脉动式阵列。

(2) 指令级并行(ILP)，在这一层我们在处理器中同时执行多个指令。例如指令流水线的使用。

(3) 线程级并行(TLP)，线程是程序的一部分，它与其他线程共享处理器资源。线程有时也被称为一个轻量级的进程。在 TLP 中，多个软件线程在一个处理器或多个处理器上同时被执行。

(4) 进程级并行。进程是一个在计算机中正在运行的程序。一个进程保留着其拥有的计算机资源，如内存空间和寄存器。当然，这是典型的多任务和分时计算，其中多个程序同时运行在共享的一台计算机或几台计算机上。

3.2　并 行 计 算

本节试图说明可用于构建并行计算机系统的不同方案。最著名的处理器分类方式是由弗林[26]基于数据及其被执行的操作而提出的：

(1) 单指令单数据流(SISD)。这是单处理器的情况。

(2) 单指令多数据流(SIMD)的。所有的处理器在不同的数据上执行相同的指令。每个处理器都在本地内存存储它自己的数据，它们之间通过典型的简单通信机制进行数据交换。许多科学和工程应用程序使用这种并行处理机制，这种应用的例子包括图形处理、视频压缩、医学影像分析等。

(3) 多指令单数据流(MISD)。神经网络和数据流机是这种并行处理器的例子。

(4) 多指令多数据流(MIMD)。每个处理器在其本地数据上运行各自的指令。这种并行处理器的例子通常来说就是多核处理器和多线程多处理器。

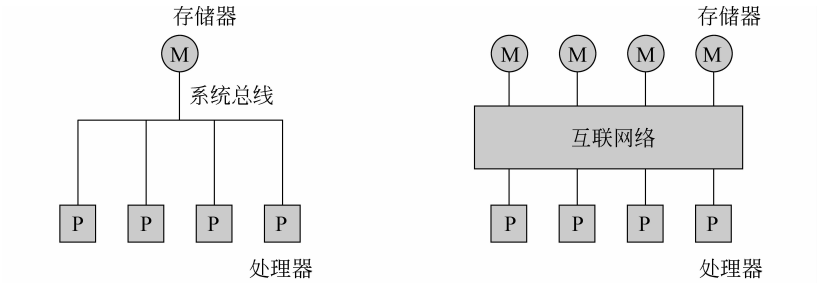
弗林的分类有点粗糙，而且我们还希望在并行计算机中更加详细地探索包括 SIMD 和 MIMD 在内的其他领域。处理器之间的同步问题并不在弗林分类标准的考虑范围内。本章将讨论最常用的并行计算机体系结构，而不是探索其他分类机制。应该指出，上述最后一种处理器类型正在迅速成为一个流行的处理系统：

- 共享内存多处理器。

- 分布式内存多处理器。
- SIMD 处理器。
- 脉动式处理器。
- 集群计算。
- 网格计算。
- 多核处理器。
- 流多处理器(SM)。

3.3 共享内存的多处理器(统一内存访问 UMA)

共享内存处理器的流行是由于其简单和通用的编程模型,它使得支持共享代码和数据的并行软件开发变得简单[27]。共享内存处理器的另一个名字是并行随机访问机(PRAM)。共享内存或共享地址空间作为处理器之间的一种通信方式。共享内存架构中的所有处理器可以通过互联网络访问一个公用内存的相同地址空间,如图 3.1(a)所示。通常情况下,这个互联网络是一种总线,但是对于更大的系统来说,网络取代了总线以提高性能。



(a) 处理器通过单一总线连接到共享内存 (b) 处理器和内存模块之间通过互联网络连接

图 3.1 共享内存多处理器架构(PRAM)

我们所说的性能是指在单位时间内进行的处理器/内存访问次数(吞吐量),以及从处理器请求内存访问到该请求被允许之间的时延(延迟)。互联网络类型及其性能分析可在文献[28]中找到。

可以直观地看到,内存带宽成为系统瓶颈,因为在一个给定的时间内只能有一个处理器访问内存。要解决这个问题,图 3.1(b)的配置用互联网络替代了总线,它允许多个处理器同时访问网络。这种配置还使用多个内存取代单个内存模块。这使得多个内存读/写操作可以同时发生。

共享内存系统以及一般并行计算机的另一个常见问题是缓存一致性,共享内存中的任何信息必须和不同 CPU 上的本地缓存中的所有副本保持一致。缓存一致性协议用于确保处理器之间的缓存一致性[20]。

在共享内存多处理器中,任何处理器可以访问任何内存模块。图 3.1(b)显示了共享内存多处理器架构。多个内存模块允许多个处理器同时访问多个内存模块。这当然增加了受互连网络限制和内存冲突影响的内存带宽。内存冲突在多于一个的处理器试图访问相同的内存模块时发生。任何内存模块设计面临的主要问题是,它通常只有一个访问端口。所以,

无论内存模块有多大，只有一个数据字节可以在任意时间内被访问。

在共享内存多处理器中，每个处理器都认为只有一个内存地址空间并且访问任何内存模块花费相同的时间。这被称为 UMA 多处理器系统。在许多共享内存多处理器中，互联网络是一个简单的总线。这就是两个以及四个奔腾处理器时的情况。

开发共享内存的多处理器并行程序不是太困难，因为所有的内存读操作对于程序员是不可见的，并且能够以串行程序一样的方式进行编写[3]。相对来说，写指令的编程更加困难，因为这个操作需要锁定数据访问直到某些线程完成对数据的处理。程序员必须识别出程序中的临界区，并引入进程间和线程间的同步机制以确保数据的完整性。诸如 POSIX 的编程库和诸如 OpenMP 的指令通过屏障、锁、监听、互斥和信号量来支持同步。

在共享内存多处理器系统中遇到的一个问题是缓存一致性。通常情况下，处理器在其自己的高速缓存中留有一个在内存模块中的数据副本。现在，如果另一个处理器改变了内存模块中这个块的内容，那么缓存中的内容就过时了。这时缓存更新政策必须被执行，以确保所有处理器中高速缓存内的副本得到更新。

同步也必须被执行，以确保多个处理器读写数据不产生冲突。信号量，互斥体和监听被用来确保数据的完整性。第 4 章对共享内存处理器有更详细的讨论。

3.4　分布式内存多处理器（非统一内存访问 NUMA）

在分布式内存多处理器中，每个内存模块和一个处理器关联在一起，如图 3.2 所示。任何处理器可以直接访问它自己的内存。消息传递（MP）机制用于允许一个处理器访问与其他处理器关联的内存模块。消息传递接口（MPI）是一种与语言无关的通信协议。

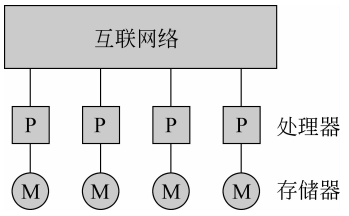


图 3.2　分布式内存多处理器体系

从这个意义上说，处理器的内存访问不是统一的，因为它取决于处理器正试图访问哪个内存模块。这被称为 NUMA 多处理器系统。

如果分布式内存多处理器是由相同的处理器组成的，则是一个对称多处理器（SMP）；如果内存的分布式多处理器是由异构的处理器组成的，则是一个非对称多处理器（ASMP）。

当分布式内存多处理器的互联网络是全球性的，如互联网，那么这个分布式内存系统通常是由成千上万台计算机集合而成，以合作解决庞大的科学问题，并且这个系统被称为不同的名字，如大规模并行计算、分布式计算、网格计算等。

3.5　SIMD 处理器

SIMD 可以被归类为一个单一程序多数据流的特殊情况（SPMD）[29]。SIMD 处理器属于共享内存多处理系统或分布式内存多处理系统。使用共享内存建立的 SIMD，适用于需要频繁交换数据的应用程序，其中一个处理器作为新的数据的生产者，许多其他的处理器作

为数据的消费者。

每个处理器与其他处理器同步执行相同的任务。正在执行的任务可能是一个简单的指令、一个线程或进程。在处理器之间分布内存可以减少内存带宽问题。

许多应用程序应用了 SIMD 处理模型，只要应用程序是可并行的。这些应用包括生物信息学、生物医学诊断、流体力学、图像处理、视频处理等。SIMD 能够显著提高应用程序的性能。有些计算机制造商在它们的处理器中加入 SIMD 扩展，并且可以运行现有的程序而不需要重新编译。同样地，一些容易学习的编程规范也利用了 SIMD 架构，例如英特尔 C++ 并行探测编译器。

适用于 SIMD 的共享内存模型的一个例子，是以下方程所描述的递归滤波器

$$y(i) = \sum_{j=0}^{N-1} [a(j)x(i-j) - b(j)y(i-j)] \tag{3.1}$$

其中 a(j)和 b(j)是滤波器系数，N 是滤波器的阶数或长度。请注意，在上面的方程中 b(0)＝0。所有的处理器实现上述方程(单指令/程序)，但作用于不同的输入数据。处理器 i 将负责生产过滤器的输出采样 y(i)并且其他 N 个处理器可能需要在各自的计算中读取这个值。

当算法的粒度很粗糙时，SIMD 机器将被称为 SPMD 机。

3.6 脉动式处理器

许多作者认为脉动式处理器属于流水线系统。而事实上，流水线处理是脉动式处理的一个特殊情况。正如我们在第 2 章中看到的，流水线是一维的，并且数据流是单向的。一个典型的管道在相邻阶段之间传输数据。脉动阵列可以是一维、二维或是三维的，如果有必要，甚至可以是更高维度的。数据沿一个或多个方向在相邻的处理器之间沿着一个或多个方向流动。在流水线系统中，每个流水线阶段执行不同的任务。在脉动式处理器中，所有的处理单元(PE)通常执行相同的任务。

一般来说，PE 之间的互连模式是从邻点到邻点的，可能还有一些全局互连。每个 PE 有一个小容量的内存来存储数据和中间结果。脉动架构适合实施数据依赖简单的高度规则的算法。这些算法包括：

(1) 线性代数，矩阵-矩阵和矩阵-向量乘法，求解线性方程组。

(2) 字符串搜索和模式匹配。

(3) 数字滤波器，例如，一维、二维和三维数字滤波器器。

(4) 在视频数据压缩中的运动估计。

(5) 有限域运算，如椭圆曲线运算。

图 3.3 显示了一个用于实现矩阵-矩阵乘法算法的简单 SIMD 处理器的例子。从图中可以看到，矩阵系数是以分布式内存的方式被存储在 PE 上的。我们也看到，处理器之间的通信是从邻点到邻点的，正如垂直箭头所示，并且使用全局布线，如水平线所示。输入数据必须是主要提供给左边缘上的处理器。输出数据从处于顶部的处理器获得。

与脉动式架构相关的设计问题有以下几种：

(1) 脉动式处理器旨在实现一个特定的算法。它必须重新设计以实现不同的算法。即使在实施相同的算法时，问题规模的改变可能需要对系统进行大量的重新设计。

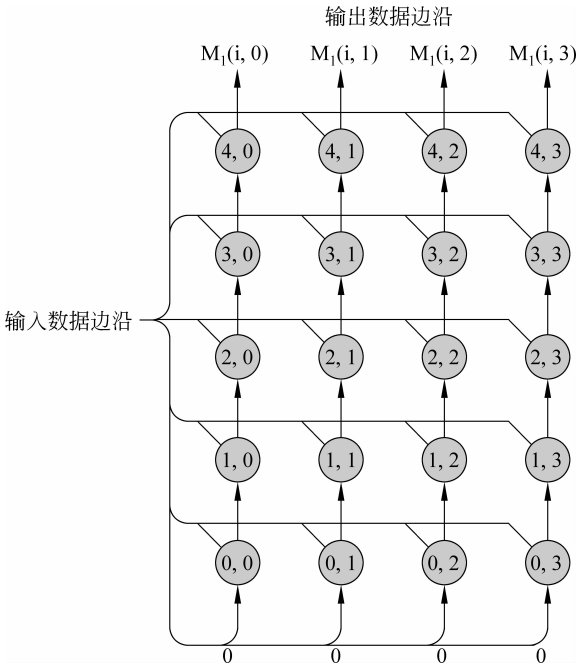


图 3.3　矩阵乘法算法的脉动式处理器

(2) 提供大量输入数据给多个处理器对系统输入/输出(I / O)的带宽是个严重的制约。在一维脉动处理器中,输入通常是先送到一个处理器再经过流水线传输到其他处理器。在其他时间,输入是通过广播总线送到 PE 或 PE 阵列的一个边缘的所有 PE。这会把脉动式处理器的性能变成受 I/O 影响的。廉价磁盘冗余阵列(RAID)可以提供一个高内存带宽的大规模存储。这个理念可以应用到闪存而非磁盘。

(3) 从多个处理器获取大量的输出数据对系统 I/O 带宽是一个严重的制约。输出可以从一个处理器,从连接所有处理器的总线,或从一个 PE 阵列的一个边缘获得。RAID 可提供高内存带宽的大规模存储。

在我们离开本节前,有必要比较一下脉动式处理器和 SIMD 处理器,因为表面上看,这两种类型的处理器都在多个数据上执行单一指令。表 3.1 从架构、内存和任务粒度相关的不同方面比较了 SIMD 和脉动阵列处理器。

表 3.1　比较 SIMD 和脉动式处理器

特征参量	SIMD	脉动阵列处理器
互联网络	任意类型	邻点到邻点和一些总线
通信模式	依赖算法	典型的邻点到邻点
处理器间通信	消息传递	简单的计时传输
处理器	简单的或复杂的	典型的非常简单的
算法应用	任意并行算法	常用迭代算法(RIA)
集成方式	独立	典型的另一系统的一部分

续表

特征参量	SIMD	脉动阵列处理器
任务粒度	典型的粗糙：进程或线程	典型的精细：简单数学操作或函数
内存	分布式	分布式和小的
布局	不可用	一维、二维、三维网格

3.7 集群计算

计算机集群是一组两个或两个以上的计算机用于执行给定问题或代码段。通常情况下，在计算集群中，将计算机连接在一起的是一个局域网络(LAN)。图 3.4 显示了一个集群计算机系统的体系结构[30]。集群中的计算机在彼此之间以及共享内存之间通信。因此，集群中处理器的通信主要通过局域网数据包。局域网通常架设在能够支持处理器之间的高速流量的服务器计算机上。共享内存必须能够在同一时间与多个处理器通信。根据共享的内存大小，它可以使用 RAID 实现。客户机在集群的处理器之间分配任务并且收集结果。

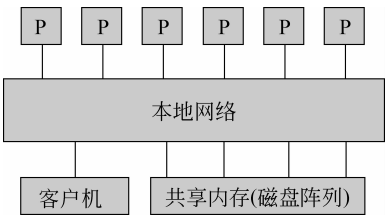


图 3.4 计算机集群系统架构

3.8 网格计算(云计算)

网格计算是指为用户提供使用分布在广域网(WAN)上的计算资源的服务。从这个意义上说，网格计算机是一个分布在广阔的地理区域的大量处理器的集合。网格计算处理的计算任务规模较大，如 N 体模拟、地震模拟、大气和海洋模拟。相对于集群计算，网格计算机是一个大的集群，其中 LAN 被诸如 Internet 的 WAN 取代。本章后面的习题总结了集群计算和云计算之间的主要区别。

一些使用云计算实现的应用包括

- 对等(P2P)计算。
- 作为服务的软件，像 Google App、Google Calendar 和 Google Mail。
- 海量存储。
- Web 应用程序和社交网络。

3.9 多核系统

多核系统通常是指所有处理器都在同一芯片上的多处理器系统。它也可以指处理器在不同的芯片上，但在同一个封装内(即一个多芯片模块)的系统。这种紧密的封装保证了较快的处理器间通信，同时保持了较低的功耗。对于双核或四核系统，处理器通信使用一条简单的总线。对于更多数量的核心，处理器使用片上网络(NOC)[13]进行互连。另一方面，多

处理器系统的处理器分布在不同的芯片上，并且处理器间互连依靠的是底板总线。继续研究并且得到一种每个芯片都是多核心芯片的多处理器系统是可能的。

多核系统的开发主要是为了提高系统的性能同时限制其功耗。换句话说，即使其组成核心是低性能处理器，多核系统仍具有良好的性能表现。相比之下，多处理器系统的开发提高了系统性能，却很少考虑功耗。一个多处理器系统具有良好的性能并且使组成的处理器也是高性能的。表 3.2 总结了多核系统与多处理器系统的主要区别。

表 3.2　多核系统和多处理器系统之间的不同之处

	多处理器系统	多核系统
集成程度	一片一个处理器	所有处理器在一个芯片上
处理器性能	高	低
系统性能	非常高	高
处理器耗能	高	低
总能耗	相对高	相对低

图 3.5 显示了多核处理器的草图。一个多核系统由以下部分组成：

(1) 通用的可编程核心。

(2) 特殊用途的加速核心。

(3) 共享内存模块。

(4) NoC(互联网络)。

(5) I/O 接口。

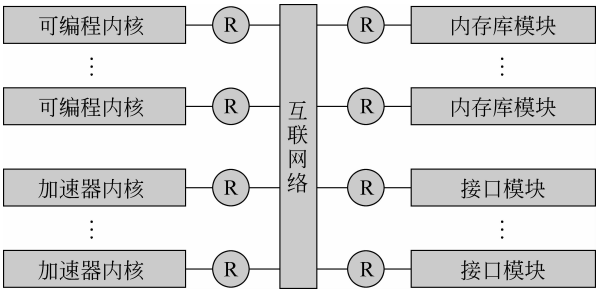


图 3.5　多核微处理器系统

为什么要走向多核系统？最主要的原因是可扩展性。当我们通过增加处理器数量来提高性能时，多核系统限制了功耗和处理器间的通信开销。多核系统还可以通过加入更多的 CPU 内核或者调整互连网络来进行扩展。要充分利用增加的资源还要做更多的工作。增加 CPU 资源数量是一回事，更好地调度它们来有效地处理任务又是另一回事了。

一些能够高效地在多核系统上实现的应用程序包括[31]：

(1) 通用多任务计算。

(2) 网络协议处理。

(3) 加密/解密处理。

(4) 图像处理。

3.10 流多处理器

流多处理器(SM)是一种 SIMD 或 MIMD 机器,其组成处理器是流处理器(SP)或线程处理器。流处理器定义为一个处理数据流的处理器,其指令集架构(ISA)包含了“核”来处理这些流[32]。流处理的概念与图形处理单元(GPU)密切相关,从而使 GPU 能够执行一般的计算密集型的通用计算。GPU 也因此成为了通用 GPU。数据流的例子有浮点数向量或视频数据处理中的一组帧像素。这种类型的数据显示出了时间和空间局部性。时间局部性是输入的数据流只使用几次来产生输出流。空间局部性是输入数据流处于内存的同一个块内。流多处理器的一个成功的例子是新一代的 GPU,如 NVIDIA 的 Fermi[33]。

适合 SM 的应用程序必须满足 3 个特点[34]:

(1) 计算密集性。

(2) 数据并行性。

(3) 消费者-生产者局部性,也就是时间和空间局部性。

计算密集性的定义是算术运算次数与 I/O 或全局内存访问次数的比值。适合流处理的应用中,这个比例可能达到 50∶1 以上。数据并行性是对输入流的所有数据并行地进行同一操作。生产者-消费者局部性是数据被读取或使用一次或若干次以产生输出流。诸如 NVIDIA 的 Fermi 的 GPU 能够支持数以万计的并行线程。

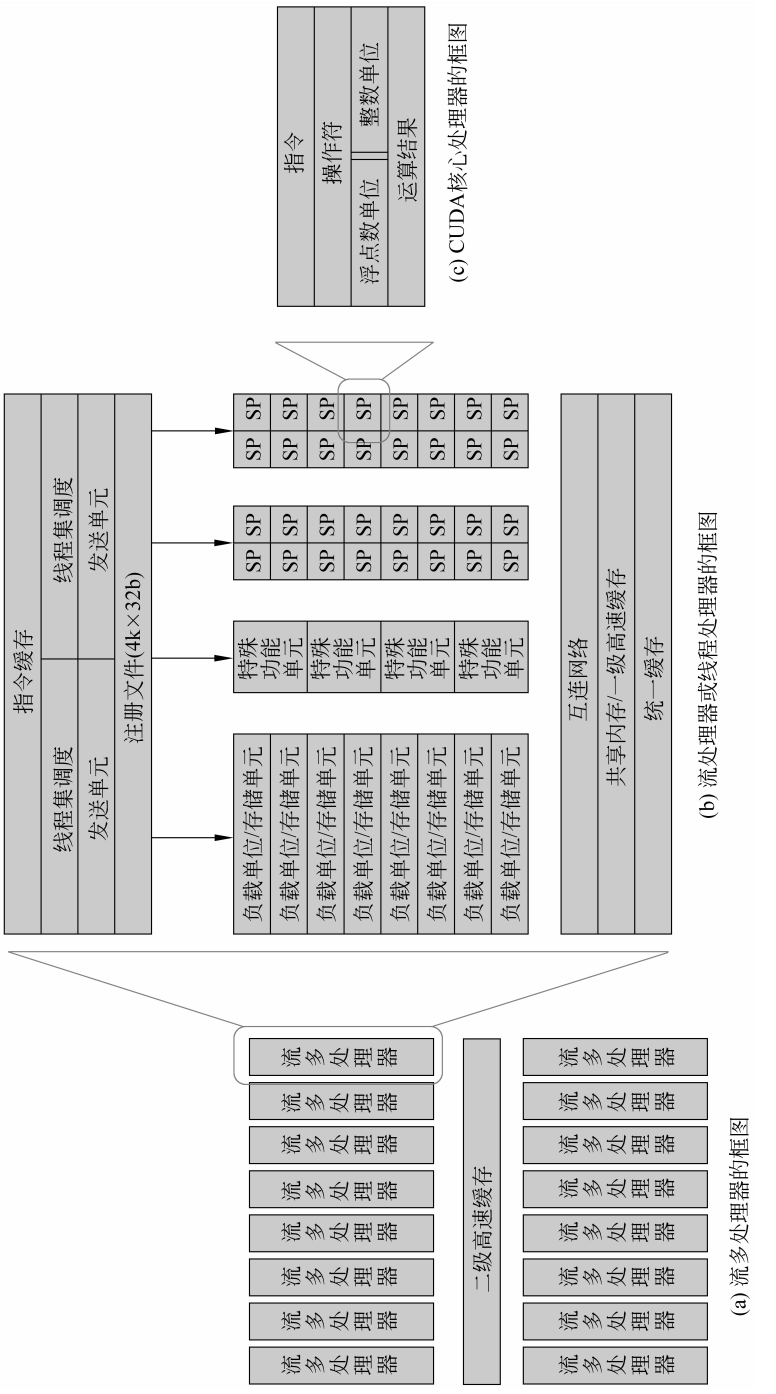
因为适合流多处理的数据显示出局限性,这些数据使用本地高速缓存,并且没有缓存命中失败。这消除了内存延时大的问题[32]。简而言之,SM 或 GPU 适合长数据序列的应用,这些数据能够使用数千个线程执行。

图 3.6 显示了一个从 NVIDIA 的费米 GPU 的框图。费米拥有 30 亿个晶体管和 512 个核心或 SP。费米能够提供高达 1.5 万亿每秒的浮点运算能力(TFLOPS)。图 3.6(a)是一个简化的费米 GPU 流多处理器。它由共享 L2 缓存的 16 个流多处理器(SM)块组成。SM 的周围是 6 个到动态随机存储器(DRAM)的 64 位接口,组成 384 位的内存位宽。

图 3.6(b)是图 3.6(a)的 16 个 SM 块之一的简化放大图。每个 SM 块包含 64 个标注着 SP 的流处理器或线程处理器,并且每四个一组,在图中表示为四列。指令到达并且被图上方标注着指令的块调度。互连网络块提供了 SM 与图底部的 L1 高速缓存之间的通信。标注着 SFU 的块是一种特殊的功能单元来计算基本函数,如在科学应用中普遍使用的平方根和三角函数。

图 3.6(c)是图 3.6(b)中的一个 SP 块的放大图。这些块被称为统一计算设备架构(CUDA)核心,拥有完整的算数逻辑单元(ALU)并能执行浮点运算。

图 3.7 比较了分配给一个 CPU 和 GPU 不同资源的比例。通用计算机有一个处理如分支预测的复杂控制的 CPU。这就是为什么 CPU 中分配给控制功能的面积是 GPU 的 8 倍。GPU 通过使用大量的缓存来存储数据以消除高速缓存命中失败和高内存延迟。在 GPU 中的 ALU 资源更多,因为 GPU 是一个流多处理器,会分配给 ALU 更多的资源。最后,DRAM 在两个系统中几乎是同样大小。



(a) 流多处理器的框图　(b) 流处理器或线程处理器的框图　(c) CUDA核心处理器的框图

图 3.6　费米 GPU 流多处理器的简化视图

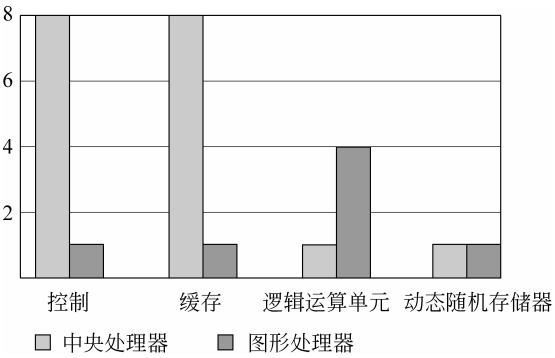


图 3.7 分配给一个 CPU 和 GPU 不同资源的比例

3.11 并行处理器之间的通信

我们在本节中回顾并行处理器如何通信以及使用什么类型的通信策略。并行处理器为了完成分配给它们的任务需要彼此之间交换数据。

3.11.1 通信类型

我们可以定义以下类型的通信模式：

(1) 一对一(单播)。

(2) 一对多(组播)。

(3) 一对全部(广播)。

(4) 收集。

(5) 规约。

图 3.8 显示了不同类型的通信模式。

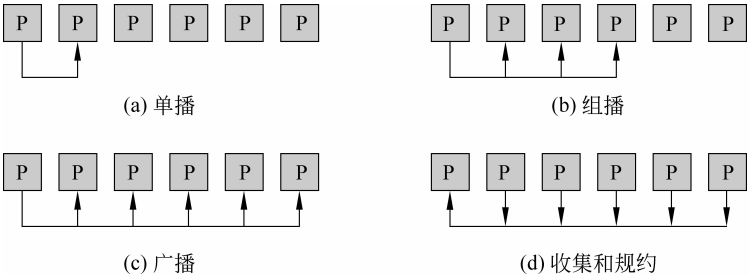


图 3.8 不同处理器间的通信模式

一对一(单播)

一对一操作涉及一对处理器：发送端和接收端。这种模式有时也称为点对点通信。我们往往在 SIMD 机中遇到这样的通信，其中每个处理器与它的邻居交换数据。图 3.8(a)显示了处理器之间的一对一通信模式。图中只显示一对处理器之间的通信，但通常情况下，所