存储器为局部存储器（local memory）。它由多线程 SIMD 处理器中的 SIMD 通道共享，但多线程 SIMD 处理器之间不共享此存储器。我们称整个 GPU 和所有线程块共享的片外 DRAM 为 GPU 存储器（GPU memory）。

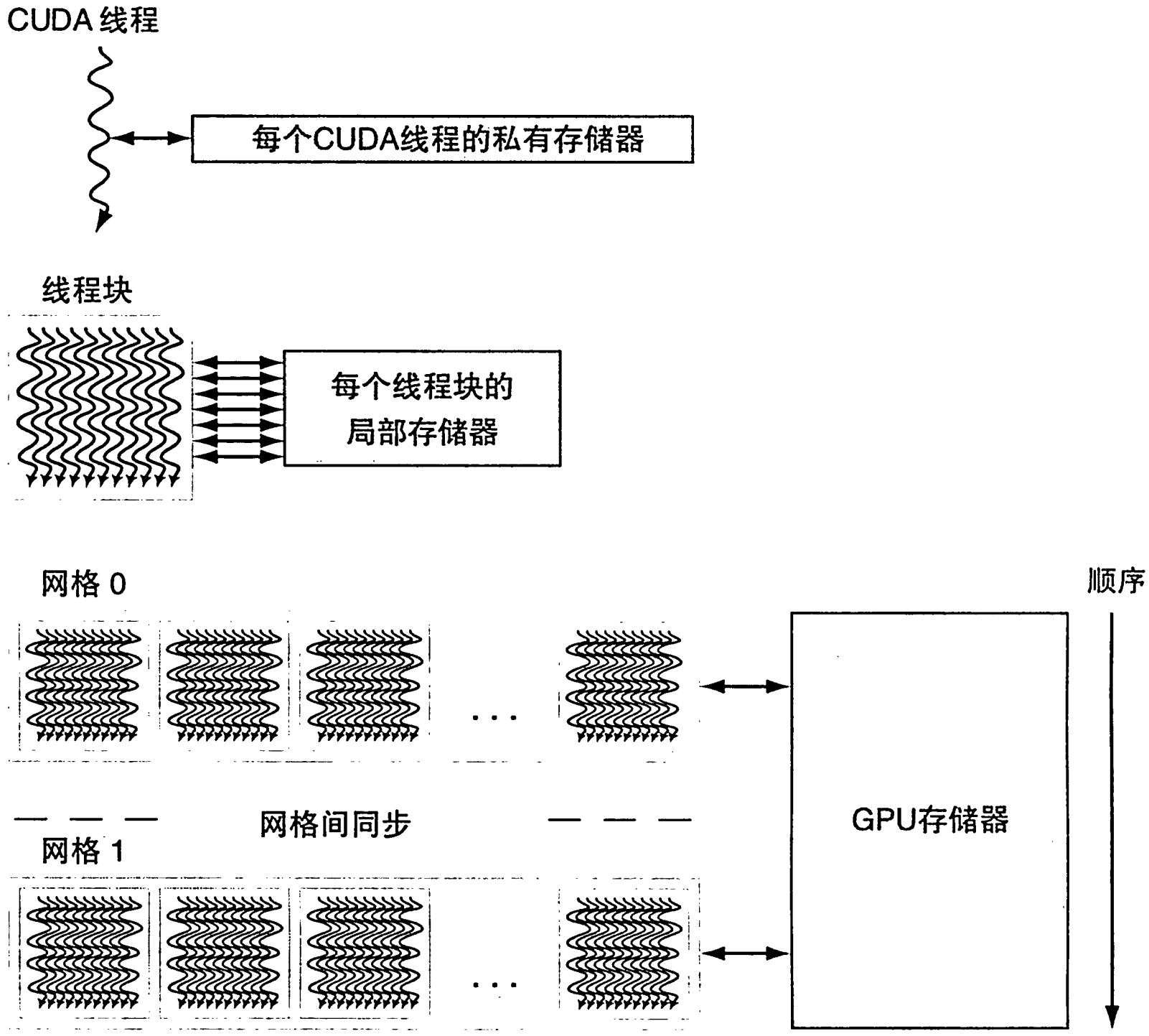


图 6-10　GPU 存储器结构。GPU 存储器被向量化的循环共享。一个线程块中的所有 SIMD 指令线程共享局部存储器

GPU 不依赖于大容量的 cache 来保存应用程序的整个工作集，而是传统地使用小容量的流 cache 并依赖于大量的 SIMD 指令多线程来隐藏 DRAM 的长延迟，因为这些工作集的大小通常是数百 MB。因此，数据集无法放在多核微处理器的末级 cache 中。为了使用硬件多线程来隐藏 DRAM 延迟，GPU 将在系统处理器中放置 cache 芯片的区域替换为计算资源和大量的寄存器，用以执行大量的 SIMD 指令多线程。

详细阐述　尽管隐藏存储器延迟是 GPU 的潜在设计哲学，但需要注意的是最新的 GPU 和向量处理器中都已经添加了 cache。举例来说，最近的 Fermi 体系结构中增加了 cache，不过它们或者被认为是带宽过滤器，以减少对 GPU 存储器的访问需求；或者作为加速器，以加速那些延迟不能被多线程隐藏的少数变量。堆栈帧、函数调用和寄存器溢出的局部存储器都很适用于 cache，因为在函数调用时延迟会有影响。cache 还可以节省能耗，因为访问片上 cache 比访问多个外部 DRAM 芯片所需的能效要低得多。

6.6.3　对 GPU 的展望

从高层次来看，具有 SIMD 指令扩展的多核计算机确实与 GPU 有相似性。图 6-11 总结了它们的相似之处和不同之处。两者都是 MIMD，并且处理器都使用多个 SIMD 通道，尽管 GPU 具有更多处理器和更多通道。两者都使用硬件多线程来提高处理器利用率，尽管

GPU 支持更多线程的硬件。两者都使用 cache，虽然 GPU 使用小容量的流 cache，而多核计算机使用大容量的多级 cache，试图完整包含整个工作集。两者都使用 64 位地址空间，尽管 GPU 中的物理地址存储器要小得多。虽然 GPU 支持页级别的内存保护，但它们还不支持动态页面调度。

特点	SIMD扩展的多核计算机	GPU
SIMD处理器数量	4～8	8～16
SIMD通道/处理器数量	2～4	8～16
支持SIMD线程的多线程硬件数量	2～4	16～32
最大cache容量	8 MiB	0.75 MiB
存储器地址大小	64位	64位
主存容量	8～256GiB	4～6GiB
页级别的内存保护	是	是
动态页面调度	是	否
cache一致性	是	否

图 6-11　带有多媒体 SIMD 扩展的多核处理器与最近的 GPU 之间的相似点与不同点

SIMD 处理器也与向量处理器很相似。GPU 中的多个 SIMD 处理器充当独立的 MIMD 核心，就像许多向量计算机具有多个向量处理器一样。按照这种观点，可以将 Fermi GTX 580 视为具有硬件支持多线程的 16 核计算机，其中每个核心具有 16 个通道。最大的区别是多线程，这是 GPU 的基础，而在大多数向量处理器中不存在。

GPU 和 CPU 体系结构没有共同的“祖先”，并没有过渡环节来解释它们。因为这种不寻常的继承关系，GPU 没有使用计算机体系结构社区中常见的术语，这导致了人们对 GPU 是什么以及 GPU 如何工作的困惑。为了解决这种困惑，图 6-12（从左到右）列出了本节中用到的更具描述性的术语、与主流计算最接近的术语、NVIDIA GPU 官方术语（如果你感兴趣的话），最后是术语的简单描述。这个“ GPU 罗塞达石”可能有助于将本节的内容和想法与更传统的 GPU 描述相关联，例如附录 B 中的描述。

虽然 GPU 正在向主流计算发展，但也不能放弃继续发展图形处理的责任。因此，当架构师提出这样的问题时，GPU 的设计可能更有意义：因为投入的硬件可以很好地完成图形处理，我们如何补充它以提高更广泛应用的性能？

在介绍了两种不同类型的具有共享地址空间的 MIMD 后，接下来我们将介绍每个处理器都有自己的专用地址空间的并行处理器，这使得构建更大的系统变得相当容易。你每天使用的因特网服务就依赖于这些大型系统。

详细阐述　虽然 GPU 中添加了与 CPU 分离的存储器，但是 AMD 和 Intel 都发布了“融合”的产品，它们结合 GPU 和 CPU 共享单个存储器。这种技术的挑战是需要在融合架构中维持高存储带宽，这也是 GPU 的基本问题。

自我检测　判断题：GPU 依靠图形 DRAM 芯片来减少访存延迟，从而提高图形应用程序的性能。

类型	更具描述性的术语	最接近的传统术语	NVIDIA GPU 官方术语	精确的定义
程序抽象	可向量化循环	可向量化循环	网格	一个可向量化的循环，在 GPU 上执行，由一个或多个可并行执行的线程块（向量化的循环体）组成
	向量化的循环体	向量化的循环（切分）体	线程块	一个在多线程 SIMD 处理器上执行的向量化的循环，由一个或多个 SIMD 指令线程组成。它们之间可以通过局部存储器进行通信
	SIMD 通道操作序列	标量循环的一次迭代	CUDA 线程	SIMD 指令线程的垂直切割，对应于由一个 SIMD 通道执行的一个元素。根据屏蔽寄存器和预测寄存器存储结果
机器对象	一个 SIMD 指令的线程	一个向量指令的线程	Warp	一个传统线程，但是包含执行在多线程 SIMD 处理器上的 SIMD 指令。根据每个元素的屏蔽寄存器存储结果
	SIMD 指令	向量指令	PTX 指令	一条横跨在多个 SIMD 通道间执行的单个 SIMD 指令
处理硬件	多线程 SIMD 处理器	（多线程的）向量处理器	流多处理器	多线程 SIMD 处理器执行 SIMD 指令线程，与其他 SIMD 处理器相互独立
	线程块调度器	标量处理器	千兆线程引擎	将多个线程块（向量化的循环体）分配到多线程 SIMD 处理器上
	SIMD 线程调度器	多线程 CPU 中的线程调度器	Warp 调度器	当 SIMD 指令线程准备好执行时调度并发射的硬件单元，包括一个追踪 SIMD 线程执行的记分板
	SIMD 通道	向量通道	线程处理器	一个 SIMD 通道，执行单个元素上的 SIMD 指令线程的操作。根据屏蔽寄存器存储结果
存储器硬件	GPU 存储器	主存	全局存储器	可以被一个 GPU 上的所有多线程 SIMD 处理器访问的 DRAM 存储器
	局部存储器	局部存储器	共享存储器	一个多线程 SIMD 处理器上的快速的局部 SRAM，不可被其他 SIMD 处理器访问
	SIMD 通道寄存器	向量通道寄存器	线程处理器寄存器	在整个线程块（向量化的循环体）中分配的一个 SIMD 通道中的寄存器

图 6-12　GPU 术语的快速介绍。第一列为硬件术语，这 12 个术语被分成了 4 组，从上到下分别是程序抽象、机器对象、处理硬件和存储器硬件

6.7　集群、仓储级计算机和其他消息传递多处理器

共享地址空间的替代方案是为每个处理器都提供私有物理地址空间。图 6-13 是这种具有私有地址空间的多处理器的典型组织结构。采用这种替代方案的多处理器必须通过显式**消息传递**（message passing）进行通信，传统上也把这种类型的计算机称为显式消息传递计算机。只要系统具有**发送消息例程**（send message routine）和**接收消息例程**（receive message routine），就可以通过内置的消息传递进行协调，因为一个处理器知道何时发送消息，并且接收处理器也知道消息何时到达。如果发送方需要确认消息已到达，则接收处理器可以向发送方发回确认消息。

迄今为止，已经有多次基于高性能消息传递网络构建大型计算

消息传递：通过显式发送和接收消息在多个处理器之间进行通信。

发送消息例程：在具有私有存储器的计算机处理器中使用的例程，用于将消息传递给另一个处理器。。

接收消息例程：在具有私有存储器的计算机处理器中使用的例程，用于接收来自另一个处理器的消息。

机的尝试了，而且这些尝试确实提供了比使用局域网构建的集群更好的通信性能。实际上，今天的许多超级计算机都使用了自定义网络。问题在于自定义网络比以太网等局域网要昂贵得多。由于过高的成本，除高性能计算之外，很少有应用程序会对其进行尝试。

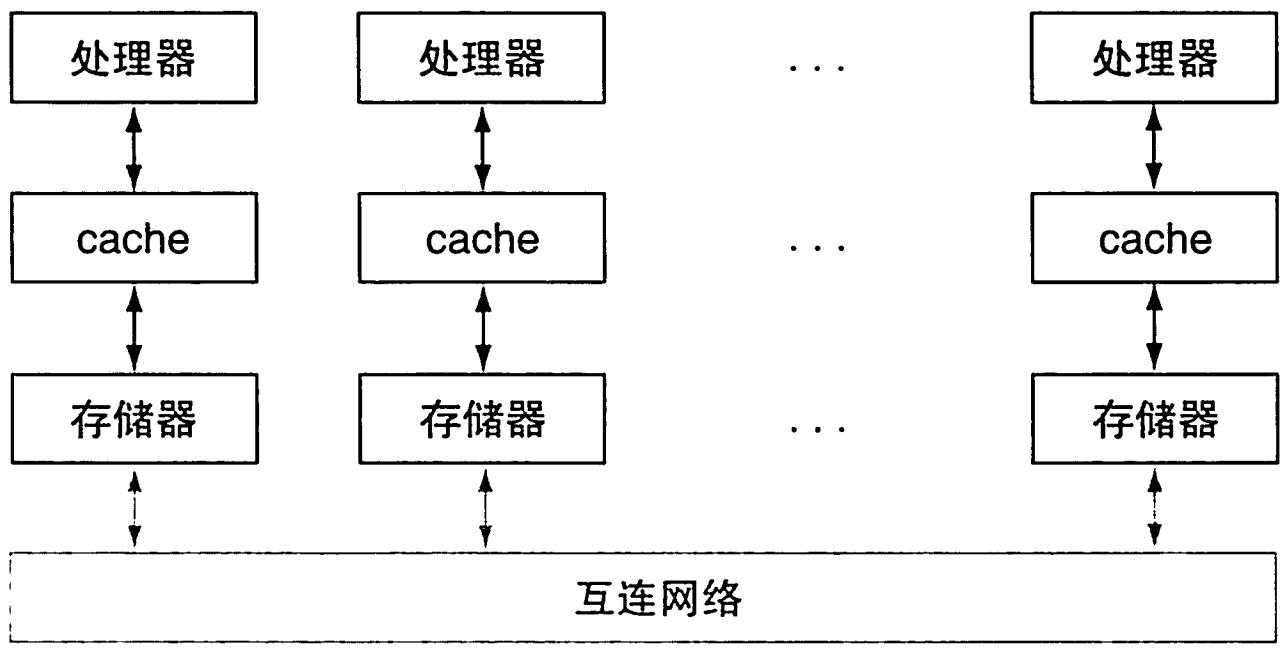


图 6-13　具有私有地址空间的多处理器的典型组织结构，传统上称为消息传递多处理器。需要注意，与图 6-7 中的 SMP 不同，本图的互连网络不在 cache 和存储器之间，而是在不同的处理器 – 存储器节点之间

硬件 / 软件接口　相对于 cache 一致的共享内存计算机，使用消息传递进行通信的计算机对硬件设计者来说更容易构建（见 5.8 节）。这种计算机对程序员来说也有好处，因为通信是显式的，所以相对于 cache 一致的共享内存计算机中的隐式通信，在性能方面会遇到的意外更少。对程序员来说，坏处是将顺序程序移植到消息传递计算机上变得更加困难，因为每次通信都必须提前定义好，否则程序将不能工作。cache 一致的共享内存计算机允许硬件来确定需要传递的数据，这使得移植更加容易。考虑到隐式通信的利与弊，对哪种方式更利于获得高性能依然存在分歧，不过现在市场上却并不存在这种困惑——多核微处理器使用共享物理内存机制进行通信，而集群节点使用消息传递机制相互通信。

集群：通过标准网络交换机上的 I/O 进行连接的计算机集合，以构成消息传递多处理器。

一些并发应用程序在并行硬件上运行良好，与硬件是否提供共享地址或消息传递机制无关。特别是，任务级并行性和几乎没有通信的应用程序——例如 Web 搜索、邮件服务器和文件服务器等——不需要共享地址机制也可以运行良好。这导致**集群**（cluster）已成为当今消息传递并行计算机中最普遍的实例。因为有单独的存储器，集群的每个节点都可以运行操作系统的不同副本。相较之下，微处理器内的核心使用芯片内部的高速网络连接，多芯片共享存储器系统使用存储器互连进行通信。存储器互连具有更高的带宽和更低的延迟，从而为共享内存多处理器提供了更好的通信性能。

将用户存储器分开存储，这种从并行编程的角度来看的弱点反而变成了系统可靠性的优点（见 5.5 节）。由于集群由通过局域网连接的独立计算机组成，因此在不关闭系统的情况下更换计算机，在集群中很容易做到，但在共享内存多处理器中却不容易。从根本上说，共享地址意味着在没有操作系统做大量工作的情况下，在服务器的物理设计上很难隔离处理器并替换它。当服务器发生故障时，集群也可以轻松地按比例缩小，从而提高可靠性。由于集群软件运行在每台计算机的本地操作系统之上，因此断开连接并更换损坏的计算机要容易得多。

考虑到集群是由完整的计算机和独立、可扩展的网络构建的，这种隔离使得在无须卸载在集群之上运行的应用程序的情况下，扩展系统更加容易。

尽管与大规模共享内存多处理器相比，集群的通信性能较差，但集群的低成本、高可用性和快速可扩展使得集群对因特网服务提供商具有吸引力。每天有数亿人使用的搜索引擎就依赖于这项技术。Amazon、Facebook、Google、Microsoft 和其他公司都有多个数据中心，每个数据中心有成千上万台服务器的集群。显然，在互联网服务公司中使用多处理器已经取得了巨大的成功。

仓储级计算机

正如上文所述，因特网服务需要建造新的建筑物，来对 100 000 台服务器进行放置、供能和冷却。虽然它们可以被归类为大型集群，不过它们的架构和操作要更为复杂。这些计算机充当一台巨型计算机，建筑施工、电费、冷却基础设施以及连接并容纳这 50 000 ～ 100 000 台服务器的网络设备的成本约为 1.5 亿美元。我们将它们认为是一类新的计算机，称为仓储级计算机（Warehouse-Scale Computer, WSC）。

> 任何人都可以构建一个高速 CPU。秘诀就是建立一个高速的系统。
> *Seymour Cray，超级计算机之父*

硬件 / 软件接口 WSC 中最流行的批处理框架是 MapReduce[Dean，2008] 及其类似的开源版本 Hadoop。受 Lisp 中的同名函数的启发，Map 首先将程序员提供的函数应用于每个逻辑输入记录。Map 在数千台服务器上运行，以产生键值对的中间结果。Reduce 收集这些分布式任务的输出，并使用另一个程序员定义的函数折叠它们。在适当的软件支持下，这两个函数都是高度并行的，易于理解和使用。只需不到 30 分钟，新手程序员就可以在数千台服务器上运行 MapReduce 任务。

例如，一个 MapReduce 程序计算大量文档中每个英语单词的出现次数。下面是该程序的简化版本，它只显示内部循环，并假设在文档中找到的所有英语单词只出现一次：

```
map(String key, String value):
      // key: document name
      // value: document contents
      for each word w in value:
      EmitIntermediate(w, "1"); // Produce list of all words
reduce(String key, Iterator values):
// key: a word
// values: a list of counts
      int result = 0;
      for each v in values:
      result += ParseInt(v); // get integer from key-value pair
      Emit(AsString(result));
```

Map 函数中使用 EmitIntermediate 函数获得文档中的每个单词和值 1。然后，Reduce 函数使用 ParseInt() 对每个文档的每个单词的所有值求和，以获得所有文档中每个单词的出现次数。MapReduce 运行时环境将 Map 任务和 Reduce 任务调度到 WSC 的服务器上。

在这种极端的规模下，需要对配电、冷却、监控和操作方面都进行创新，WSC 是 20 世纪 70 年代超级计算机的现代版本，这使得 Seymour Cray 成为当今 WSC 体系结构的教父。他创造的极致计算机可以完成其他计算机不能完成的计算，但是价格十分昂贵，只有少数公司能负担得起。现在 WSC 的目标是为世界提供信息技术，而不是为科学家和工程师提供高性能计算。因此，今天的 WSC 肯定比过去 Cray 的超级计算机发挥了更重要的社会作用。

尽管在目标上与服务器有一些共同点，但 WSC 还是有以下三个主要区别：

1. 大量、简单的并行性：服务器架构师关注的是目标市场中的应用程序是否具有足够的并行性，以此来判断并行硬件的数量，以及成本是否太高以至于没有足够的通信硬件来利用这种并行性。而WSC架构师没有这样的担忧。首先，像MapReduce这样的批处理应用程序受益于需要独立处理的大量独立数据集，例如Web爬虫得到的数十亿个Web页面。其次，交互式互联网服务应用程序，也称为**软件即服务**（Software as a Service，SaaS），可以从其数百万个独立用户中受益。读取和写入很少依赖于SaaS，因此SaaS很少需要同步。例如，搜索使用只读索引，电子邮件通常是读写独立信息。我们称这种类型的简单并行为请求级并行，因为许多独立的工作可以自然地并行进行，几乎不需要通信或同步。

软件即服务：软件不是销售并安装运行在用户计算机上，而是在远程站点运行，并通过Internet（通常通过Web界面）提供给客户。SaaS客户是根据是否使用而不是是否拥有收费的。

2. 计算运营成本：传统上，服务器架构师在成本预算内设计其系统以实现最佳性能，并且只关心能耗以确保它们不超过其机箱的冷却能力。他们通常会忽略服务器的运营成本，并假设运营成本与购买成本相比显得微不足道。WSC有更长的使用寿命——建筑、电气和制冷基础设施通常会在10年或更长的时间内平摊开销——因此运营成本需要将以下内容加起来：对于运营超过10年的WSC，能耗、配电和冷却的成本占据总成本的30%以上。

3. 规模及与规模相关的机会和问题：要构建一个WSC，你必须购买100000台服务器以及基础架构，而这也意味着批量折扣。因此，WSC内部结构如此庞大，即便WSC的总数很少也可以获得规模经济。这些规模经济导致了云计算，因为WSC的服务器单位成本较低，意味着云公司可以用有利可图的价格租用服务器，并且这个价格仍然低于外部人员自己购买的成本。规模经济机会的另一面是需要应对大规模的故障率。即使服务器具有高达25年的（200000小时）的平均无故障时间，但WSC架构师仍需要考虑每天有五台服务器故障的可能。5.15节提到的年化磁盘故障率（AFR）在谷歌的测量结果为2%～4%。如果每台服务器有四个磁盘且其年度故障率为2%，那么WSC架构师可以估算出平均每小时可能有一个磁盘发生故障。因此，相比服务器架构师来说，对于WSC架构师而言容错性更为重要。

由WSC带来的规模经济实现了长期以来希望将计算作为一种实用工具的目标。云计算意味着任何有好主意、商业模式和信用卡的人都可以利用数千台服务器在世界各地即时传递他们的愿景。当然，也存在可能会限制云计算增长的重要障碍，例如安全性、隐私、标准和互联网带宽的增长率，但我们预见这些问题可以得到解决，而WSC和云计算能够因此蓬勃发展。

考虑到云计算的增长速度，2012年亚马逊网络服务公司宣布，它每天都会增加相当于2003年服务器总量的新服务器来支持亚马逊的所有全球基础设施，当时亚马逊是一家拥有6000名员工的年收入达52亿美元的企业。

现在我们已经了解了消息传递多处理器的重要性，特别是对于云计算，接下来要介绍将WSC的节点连接在一起的方法。由于摩尔定律和每个芯片的核心数量不断增加，现在芯片内部也需要互连网络，因此这些拓扑结构无论是在小型计算机还是大型计算机中都很重要。

详细阐述 MapReduce框架在Map阶段结束时对键值对进行重组和排序，以生成所有共享相同键的组。接下来将这些组传递到Reduce阶段。

详细阐述 还有一种形式的大规模计算是网格计算，其中计算机分布在一个很大的区域内，运行于其上的程序必须通过长距离网络进行通信。最受欢迎也最独特的网格计算形式由

SETI@home 项目创立。由于在任何时候都存在数以百万计的 PC 闲置无用，如果有人开发出可以在这些计算机上运行的软件然后分发到每台 PC 上独立解决问题，那么问题的结果就可以被获取并充分利用。第一个例子是 1999 年在加州大学伯克利分校推出的搜索超级地球智能（SETI）。超过 200 个国家的 500 多万计算机用户报名参加了 SETI@home，其中超过 50% 的用户在美国境外。截至 2011 年底，SETI@home 网格的平均性能为 3.5PetaFLOPS。

自我检测

判断题：与 SMP 一样，消息传递计算机依赖锁来进行同步。

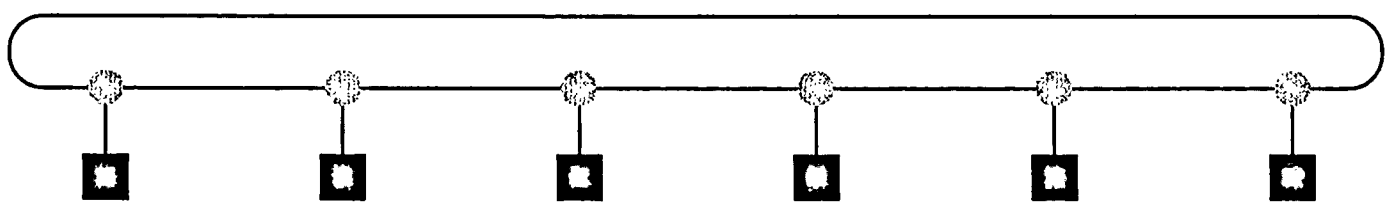
判断题：集群具有单独的存储器，因此需要许多操作系统的副本。

6.8　多处理器网络拓扑简介

多核芯片需要片上网络将核心连接在一起，而集群需要局域网将服务器连接在一起。本节将对不同互连网络拓扑的优缺点进行回顾。

网络成本包括开关数量、开关连接到网络的链路数量、每个链路的宽度（比特数），以及网络映射到芯片时连接的长度。例如，一些核心或服务器可以是相邻的，而其他核心或服务器则可能位于芯片的另一侧或数据中心的另一侧。网络性能也是多方面的，包括在无负载的网络上发送和接收消息的延迟、在给定时间段内可以传输的最大消息数量的吞吐量、由网络的一部分争用引起的延迟，以及取决于通信模式的可变性能。网络的另一个责任是容错，因为系统可能需要在存在损坏组件的情况下工作。最后，在受能耗限制的系统里，由于不同组织架构导致的系统能耗的不同可能是需要考虑的最重要的问题。

网络通常被绘制为图形，图形的每条边代表通信网络的链路。在本节的图中，处理器 – 内存节点显示为黑色方块，开关显示为灰色圆点。我们假设所有链接都是双向的；也就是说，信息可以向任一方向流动。所有网络都由开关组成，其链路连接到处理器 – 内存节点和其他开关。第一个网络将一系列节点连接在一起：



这种拓扑称为环。由于某些节点没有直接连接，因此某些消息必须沿中间节点跳转，直到到达最终目的地。

与总线（一组允许广播到所有连接设备的公共线路）不同，环能够同时进行多个传输。

由于有多种拓扑可供选择，因此需要使用性能指标来区分这些设计。有两个很受欢迎的指标。第一个是总**网络带宽**，即每个链路的带宽乘以链路数量，代表峰值带宽。对于上述的环拓扑，如果有 P 个处理器，总网络带宽将是一条链路带宽的 P 倍；总线的总网络带宽就是该总线的带宽。

网络带宽：非正式术语，指网络的峰值传输速率；可以指单个链路的速率，或网络中所有链路的集体传输速率。

为了避免只评价最佳带宽的情况，我们提供了另一个更接近最坏情况的指标：**二分带宽**。通过将机器分成两半来计算该度量，然后将跨越假想分界线的链接的带宽相加。环的二分带宽是链路带宽的两倍，是总线链路带宽的一倍。如果单个链路与总线一样快，则

二分带宽：多处理器的两个相等部分之间的带宽。此度量适用于多处理器的最坏情况分割。

在最坏的情况下，环路的速度是总线的两倍，在最好的情况下，它的速度是总线的 P 倍。

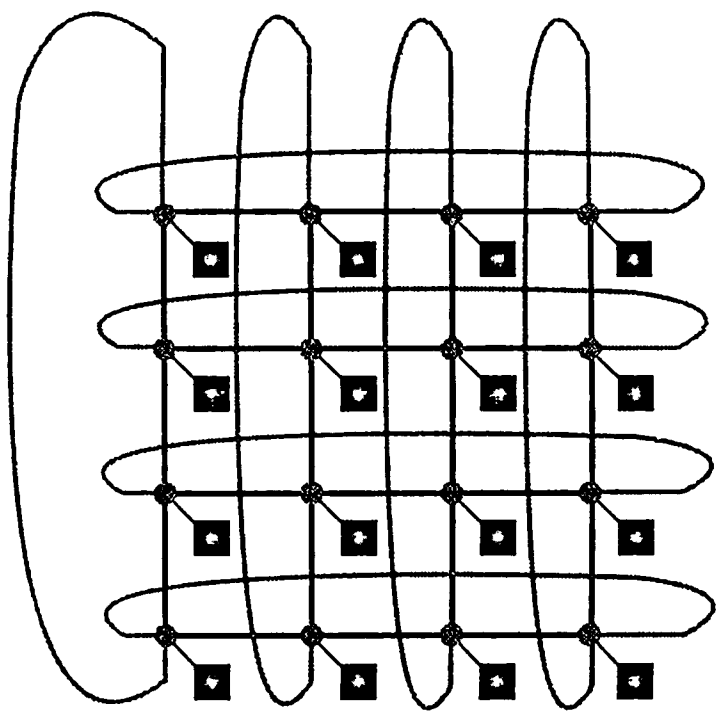
由于某些网络拓扑不是对称的，因此产生的问题是二分机器时在哪里绘制假想分界线。由于二分带宽是最坏情况的度量，因此答案是选择可以产生最差网络性能的分区。换言之，就是计算所有可能的二分带宽并选择其中最小的一个。我们之所以采取这种悲观的观点，是因为并行程序通常会受到通信链中最薄弱链路的限制。

环的另一个极端是**全连接网络**，其中的每个处理器都具有到其他处理器的双向链路。对于全连接网络，总网络带宽为 $P\times(P-1)/2$，二分带宽为 $(P/2)^2$。

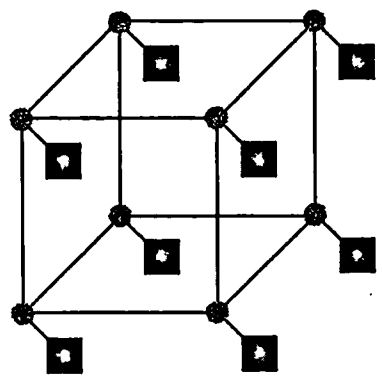
全连接网络：通过在每个节点之间提供专用通信链路来连接处理器－内存节点的网络。

全连接网络对性能的巨大提升被成本的急剧增长所抵消。这种结果激励工程师去发明新的拓扑结构，使其既有环的成本又有全连接网络的性能。对结构的评估在很大程度上取决于在计算机上运行的并行程序工作负载的通信特点。

在公开出版物中已经讨论过的不同拓扑结构的数量难以估计，但在商用并行处理器中只使用了少数的拓扑结构。图 6-14 中给出了两种流行的拓扑结构。



a）2D网格/16个节点的网格



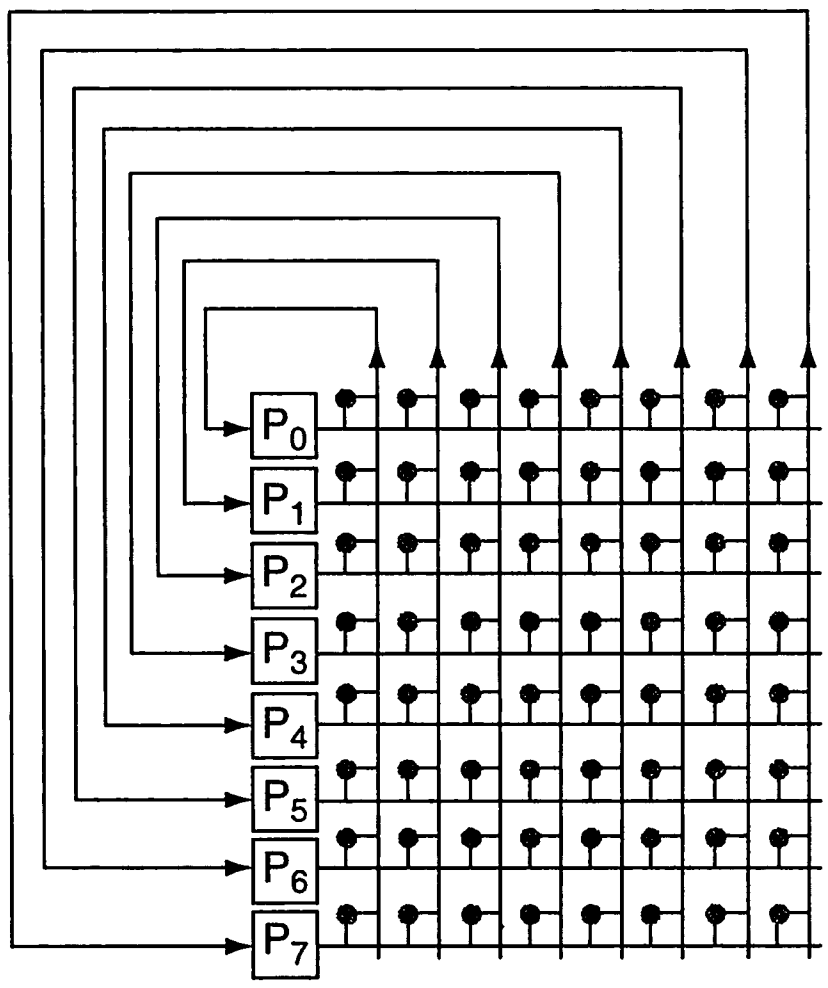
b）8个节点的n立方树（$8=2^3$，因此$n=3$）

图 6-14　出现在商用并行处理器中的网络拓扑。灰色圆点表示开关，黑色方块表示处理器－内存节点。尽管一个开关有很多链接，但通常只有一个链接连接到处理器。布尔 n 立方体拓扑是具有 2^n 个节点的 n 维互连，每个开关需要 n 个链路（再加上一条连接处理器的链路），因此存在 n 个最近邻节点。这些基本拓扑结构通常会补充额外的链路，以提高性能和可靠性

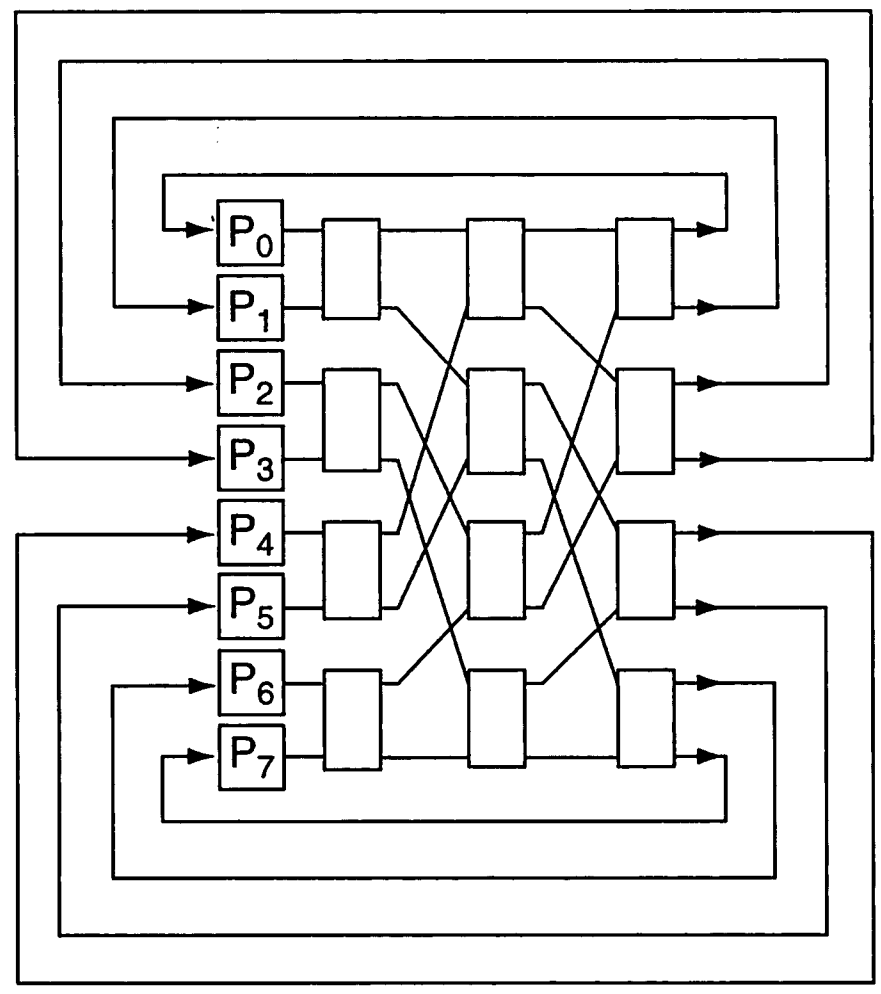
将处理器放置在网络中的每个节点处的替代方案是：仅在这些节点中的部分节点处留下小于处理器－内存－开关节点的开关，这种开关规模更小，因此可以更密集地打包，从而减小距离并提高性能。这种网络通常被称为**多级网络**（multistage network），以反映消息可能途经多个步骤后才能传播。多级网络的类型与单级网络一样多，图 6-15 是两个主流的多级网络组织结构。**全连接**（fully connected）或**交叉开关网络**（crossbar network）允许任何节点在通过网络时可与任何其他节点通信。Omega 网络所使用的硬件少于交叉开关网络（前者需要 $2n\log_2 n$ 个开关，后者需要 n^2 个开关），但消息之间可能会发生争用，具体取决于通信模式。例如，图 6-15 中的 Omega 网络在从 P_1 发送消息到 P_4 的同时，无法从 P_0 向 P_6 发送消息。

多级网络：在每个节点上提供小型开关的网络。

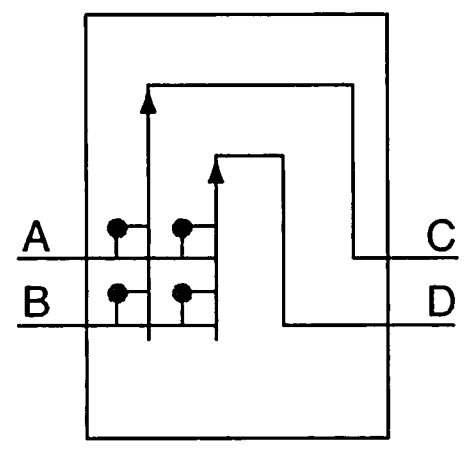
交叉开关网络：允许任何节点在通过网络时与任何其他节点通信的网络。



a）交叉开关网络



b）Omega网络



c）Omega网络的开关盒

图 6-15　8 个节点的主流多级网络拓扑。这些图中的开关比之前图中的开关更简单，因为链路是单向的；数据从左侧进入并从右侧退出链接。图 c 中的开关盒可以将 A 传送到 C、将 B 传送到 D；或将 B 传送到 C、将 A 传送到 D。交叉开关网络使用 n^2 个开关，其中 n 是处理器的数量，而 Omega 网络使用 $2n \log_2 n$ 个大型开关盒，每个开关盒逻辑上由四个较小的开关组成。在这种情况下，交叉开关网络使用 64 个开关，而 Omega 网络使用 48 个开关，或者 12 个开关盒。但是，交叉开关网络可以支持处理器之间的任何消息组合，而 Omega 网络则不能

网络拓扑的实现

对本节中提到的所有网络，我们只做了简单的分析，而忽略了构建网络时需要考虑的重要实际因素。每个链接之间的距离会影响高时钟频率下通信的成本——通常，距离越长，以高时钟频率运行的成本越高。较短的距离也会使得为链接分配更多的电线变得更容易，因为如果电线较短，那么驱动这些电线所需要的功耗就更小，而且较短的电线也比较长的电线更便宜。另一个实际的限制是：必须将三维的图像绘制到本质上其实是二维介质的芯片上。最后要关注的是能耗，例如，由于能耗的限制可能会迫使多核芯片更依赖于简单的网络拓扑。最重要的是，当使用硅片或数据中心进行构造时，在黑板上勾勒出来的看起来优雅的拓扑结构在真正制造时可能是不切实际的。

现在我们已经了解了集群的重要性，并且学习了可以遵循的拓扑结构，接下来将会看到网络与处理器的软硬件接口。

自我检测　判断题：对一个具有 P 个节点的环，总网络带宽与二分带宽的比率为 $P/2$。

6.9　与外界通信：集群网络

本节是网络章节，描述了用于将集群节点连接在一起的网络硬件和软件。示例中使用 PCIe 连接到计算机的万兆以太网上。该示例显示了如何优化软硬件以提高网络性能，包括零拷贝消息传递、用户空间通信、使用轮询而不是 I/O 中断，以及硬件计算校验和。虽然示例是互连网络，但本节中的技术也适用于存储控制器和其他 I/O 设备。

本节从底层详细介绍了网络性能，下一节将从更高的层次介绍如何对多处理器进行基准测试。

6.10　多处理器测试基准和性能模型

在第 1 章中我们看到，基准测试系统一直是一个敏感话题，因为它是判断哪个系统更好的一种很直观的方式。测试结果不仅影响商业系统的销售，还影响系统设计者的声誉。因此，每个参与者都希望自己获胜，但如果其他人获胜，他们也希望获胜者实至名归，因为他们的系统是真的好。这些期望导致测试结果不能只是针对测试基准程序的简单伎俩，而应该是能够真正促进实际应用程序的性能提高。

为了避免可能的作弊行为，一个典型的原则是不能修改基准测试程序。源代码和数据集是固定的，并且只有唯一的正确结果。对这些原则的任何违反都会导致测试结果无效。

许多多处理器基准测试程序都遵循这些规则。一个共同的例外是允许扩大问题规模，这样可以在有不同数量处理器的系统上运行基准测试程序。也就是说，许多基准测试程序允许弱比例缩放而不是强比例缩放。但即便如此，在比较不同问题规模的测试结果时仍要小心。

图 6-16 给出了几个并行基准测试程序的总结，下文将做具体描述。

- Linpack 是一组线性代数例程，这些例程执行高斯消元。3.5.6 节例题中的 DGEMM 例程就是 Linpack 基准测试程序中的一小部分源代码，但它占据了该基准测试的大部分执行时间。它允许弱比例缩放，让用户选择任何规模的问题。而且，只要保证计算结果正确并对相同规模的问题执行相同数量的浮点运算。Linpack 允许用户以几乎任何形式和任何语言重写 Linpack。计算 Linpack 最快的 500 台计算机每年在 www.top500.org 发布两次。排名第一的计算机被新闻界视为世界上最快的计算机。
- SPECrate 是基于 SPEC CPU 基准测试（如 SPEC CPU 2006，见第 1 章）的吞吐量指标。SPECrate 并不报告各个程序的性能，而是同时运行该程序的很多副本。因此，它测量的是任务级并行性，因为这些任务之间没有通信。而且可以根据需要运行任意数量的程序副本，这也是一种弱比例缩放的形式。
- SPLASH 和 SPLASH 2（Stanford Parallel Applications for Shared Memory）是 20 世纪 90 年代斯坦福大学的研究成果，目的是提供类似于 SPEC CPU 的并行基准测试程序。它由核心程序和应用程序组成，许多来自高性能计算领域。尽管该程序提供了两组数据集，但仍需要强比例缩放。
- NAS（NASA Advanced Supercomputing）并行基准测试是 20 世纪 90 年代对多处理器基准测试程序的另一尝试，由 5 个来源于流体动力学的核心程序构成，允许通过定义几个数据集实现弱比例缩放。像 Linpack 一样，这些基准测试程序可以被重写，但编程语言只能使用 C 或 Fortran。
- 最近的 PARSEC（Princeton Application Repository for Shared Memory Computer）基准测试程序集由 Pthread（POSIX 线程）和 OpenMP（Open MultiProcessing，见 6.5 节）的多线程程序组成。它们主要专注于新兴的计算领域，由 9 个应用程序和 3 个核心程序构成。其中 8 个依赖于数据并行，3 个依赖于流水线并行，另一个依赖于非结构化并行。

  PThread：创建和操作线程的一个 UNIX API。它被组织成一个库的形式。
- 在云端，Yahho! Cloud Serving Benchmark（YCSB）的目标是比较云数据服务的性能。

它通过使用 Cassandra 和 HBase 作为具有代表性的例子，提供了一个易于让用户评测新数据服务的框架 [Cooper，2010]。

- 加州大学伯克利分校的研究人员提出了一种方法。他们确定了 13 种面向未来应用程序的设计模式。这些设计模式使用框架或核心实现，一些实例包括稀疏矩阵、结构化网格、有限状态自动机、MapReduce 和图遍历等。通过将定义保持在高级别层次，他们希望鼓励在系统的任何层次进行创新。因此，速度最快的稀疏矩阵求解器的系统除了使用新型体系结构和编译器之外，还可以使用任何数据结构、算法和编程语言。

基准测试程序	可扩展性	可编程性	描述
Linpack	弱	是	稠密矩阵线性代数 [Dongarra，1979]
SPECrate	弱	否	独立任务并行化 [Henning，2007]
面向共享存储器的斯坦福并行应用SPLASH2 [Woo et al., 1995]	强（虽然提供了两种问题规模）	否	复杂1D FFT 模块化LU分解 模块化稀疏丘拉斯基分解 整数基数排序 巴尔内斯小屋 适应性快速多极算法 海洋仿真 光线跟踪 声音渲染器 空间数据结构的水仿真 非空间数据结构的水仿真
NAS并行基准测试程序 [Bailey et al., 1991]	弱	是（只能是C或Fortran）	EP：令人尴尬的并行内核 MG：简化的多重网格计算 CG：面向共轭梯度方法的非结构化网格 FT：使用FFT的3D偏微积分方程 IS：大型整数排序
PARSEC 基准测试程序集 [Bienia et al., 2008]	弱	否	Blackscholes：使用毕苏期权定价模式的期权定价 Bodytrack：人体跟踪 Canneal：使用cache感知的模拟退火进行布线优化 Dedup：采用数据去重的下一代压缩 Facesim：人脸运动仿真 Ferret：内容相似性搜索服务器 Fluidanimate：SPH方法的流体动力学动画 Freqmine：常见物品集合的数据挖掘 Streamcluster：输入流的在线分类 Swaptions：交换组合的定价 Vips：图像处理 X264：H.264视频编码
伯克利设计数据集 [Asanovic et al., 2006]	强或弱	是	有限状态自动机 组合逻辑 图的遍历 结构化网格 稠密矩阵 稀疏矩阵 波谱法 动态程序设计 N体问题 MapReduce 反向跟踪/分支与边界 图模型推导 非结构化网格

图 6-16　并行基准测试程序的实例

基准测试程序原有约束所造成的负面影响是创新被局限到体系结构和编译器中。更好的数据结构、算法、编程语言等通常不能被使用，因为这些可能产生容易令人误解的结果。这样系统可能因为算法而获胜，而不是因为硬件或编译器。

虽然这些准则在计算基础相对稳定时是可以理解的——因为它们是在 20 世纪 90 年代提出的，而且是在 90 年代的前 5 年——但是，这些准则在编程变革中就不合时宜了。要使变革成功，我们需要鼓励各个层次的创新。

6.10.1　性能模型

与基准测试程序相关的一个话题是性能模型。正如我们在本章中看到越来越多的体系结构多样性——多线程、SIMD、GPU——如果我们有一个简单的模型来分析不同体系结构设计的性能，将是十分有益的。这个模型不一定是完美的，只要有所见地就行。

第 5 章中用于分析 cache 性能的 3C 模型是性能模型的一个例子。它不是一个完美的性能模型，因为它忽略了块大小、块分配策略和块替换策略等潜在的重要因素。而且，它还存在一些含糊其辞的地方。例如，一次失效在一种设计中可以归为容量失效，而在同样容量的另一个 cache 中可以归为冲突失效。然而，3C 模型已经流行了 25 年，因为它提供了对程序行为的深入理解，有助于体系结构设计者和程序员根据对该模型的观察改进他们的创新成果。

为了找到并行计算机的这种模型，让我们从小的核心程序开始，如图 6-16 的 13 个 Berkeley 设计模式。尽管这些核心程序的不同数据类型有许多版本，但浮点在几种实现中很常见。因此，在给定的计算机上峰值浮点性能是这类核心程序的速度瓶颈。对于多核芯片，峰值浮点性能是芯片上所有处理器核峰值性能的总和。如果系统中包含多个处理器，那么应该将每个芯片的峰值性能与芯片总数相乘。

对存储器系统的需求可以用峰值浮点性能除以每访问一字节所包含浮点操作数的平均值来估算：

$$\frac{\text{浮点操作数/秒}}{\text{浮点操作数/字节}} = \text{字节/秒}$$

存储器每访问一字节所包含的浮点运算比例称为**算术强度**（arithmetic intensity）。它的计算可以用程序中浮点运算的总数除以程序执行期间内存传输数据的总字节数。图 6-17 给出了图 6-16 中几种 Berkeley 设计模式的算术强度。

算术强度：一个程序中浮点操作数量与访问内存字节数量的比值。

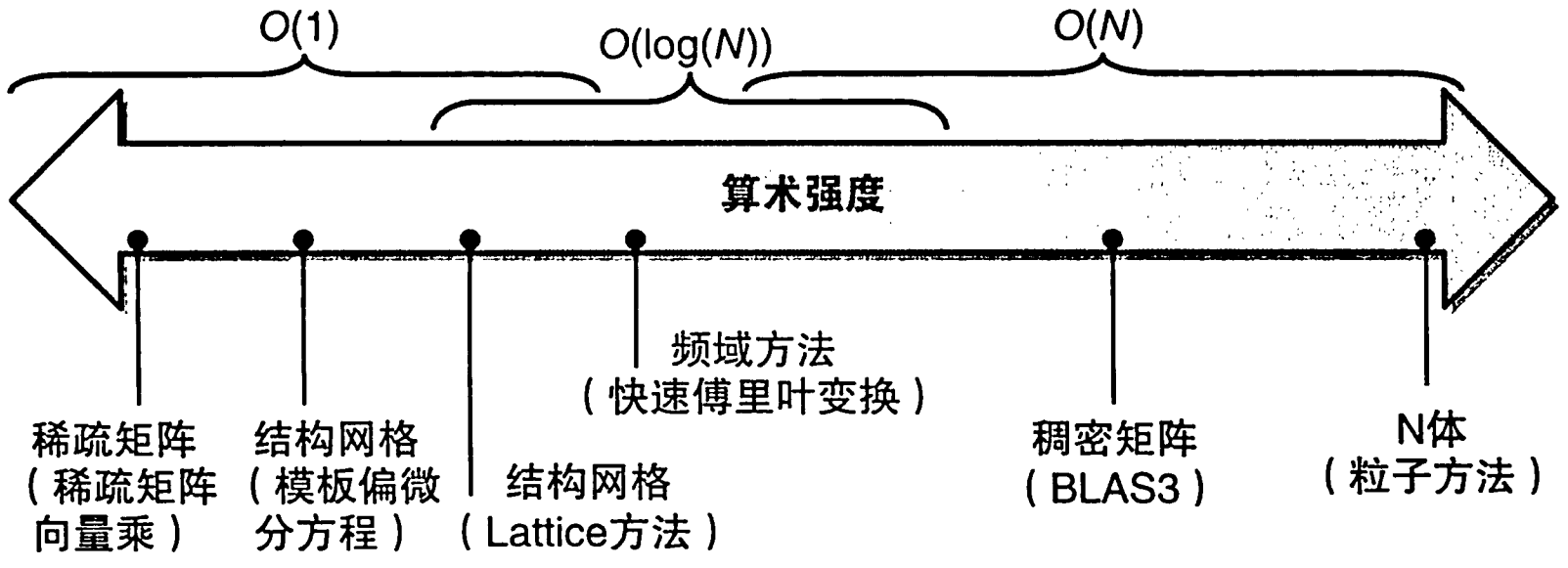


图 6-17　算术强度，计算方式为运行程序中的浮点运算总数除以访问内存的总字节数[Williams，Waterman，and Patterson，2009]。一些核心程序的算术强度与问题规模成比例扩展，例如密集矩阵，但是也有许多核心程序的算术强度与问题规模无关。对于前者，弱比例缩放可能会导致不同的结果，因为它对存储系统的需求不是很大

6.10.2 Roofline 模型

本节提出的简单模型将浮点性能、算术强度和内存性能结合在一张二维图中 [Williams, Waterman, and Patterson, 2009]。峰值浮点性能可以在上面谈到的硬件规格说明书中找到。我们在这里考虑的核心程序的工作集不适合使用片上 cache，因此峰值内存性能可以由 cache 后面的存储系统定义。获取峰值内存性能的一种方法是使用流式基准测试程序（见 5.2.2 节的“详细阐述”）。

图 6-18 给出了针对一台计算机的模型，注意不是针对每个核心程序的模型。*Y* 轴表示浮点性能，范围是 0.5 ～ 64.0GFLOP/s。*X* 轴表示算术强度，范围是 1/8FLOP/DRAM 字节～ 16 FLOP/DRAM 字节。注意该图是 log-log 比例。

对于给定的核心程序，我们可以根据其算术强度在 *X* 轴上找到对应点。如果我们在该点绘制一条垂直线，那么该核心程序在计算机上的性能一定在该垂直线的某个位置上。我们可以画一条水平线表示该计算机的峰值浮点性能。显然，实际的浮点性能不会超过该水平线，因为这是一个硬件上限。

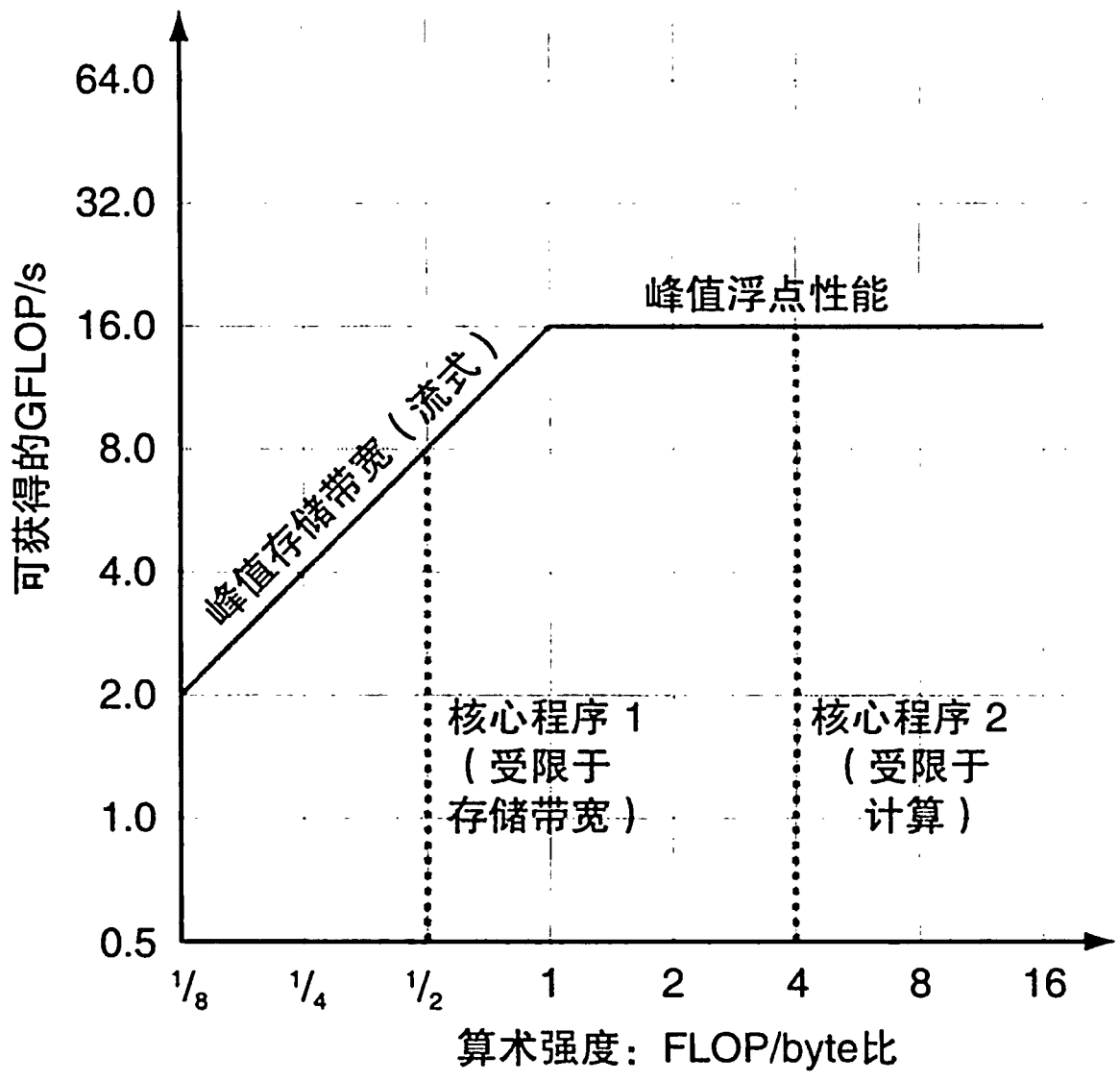


图 6-18　Roofline 模型 [Williams, Waterman, and Patterson, 2009]。本例具有 16GFLOP/s 的峰值浮点性能和 16GB/s 的峰值内存带宽，该数据来自流式测试程序。（由于实际上是四次测量，图中的线是四次的平均值。）左边的虚线表示核心程序 1，其算术强度为 0.5FLOP/byte。在 Opteron X2 上，受限于低于 8GFLOP/s 的内存带宽。右边的虚垂直线表示核心程序 2，其算术强度为 4FLOP/byte。它只受限于 16GFLOP/s 的计算（该数据基于 AMD Opteron X2（版本 F），使用双 socket 系统中的 2GHz 的双核）

我们如何画出峰值内存性能呢（以 byte/s 为单位）？由于 *X* 轴是 FLOP/byte，*Y* 轴是 FLOP/s，因此 byte/s 只是图中一条 45° 的对角线。因此，我们可以绘制出第三条线来表示对于给定的算术强度，该计算机存储系统所能支持的最大浮点性能。我们可以用下面的公式表示该界限，以便在图 6-18 中画出该线：

可达到的 GFLOP/s = min（峰值存储带宽 × 算术强度，峰值浮点性能）

水平线和对角线给出了简单模型的名称并标出了对应值。这个像屋顶一样的轮廓线设定了一个核心程序在不同算术强度下的性能上界。给定一个计算机的 Roofline 模型，你可以重复地使用它，因为它不会随核心程序而变化。

如果我们认为算术强度是支撑屋顶的一根杆，那么它要么支撑屋顶的平坦部分，这表示性能受计算限制；要么支撑屋顶的倾斜部分，这表示性能受存储器带宽限制。在图 6-18 中，核心程序 1 属于前者，而核心程序 2 属于后者。

需要注意的是“脊点”，它是屋顶平坦部分与倾斜部分的交叉点，这对计算机来说是一个关键点。如果它过于靠右，那么只有极高算术强度的核心程序才能获得最大性能。如果它过于靠左，那么几乎所有核心程序都可以达到最大性能。

6.10.3　两代 Opteron 的比较

四核 AMD Opteron X4（Barcelona）是两核 Opteron X2 的后续版本。为了简化主板设计，它们使用了相同的插座。因此，它们具有相同的 DRAM 通道，也就具有相同的峰值存储带宽。除了将核心程序数量加倍之外，Opteron X4 还将每个核的峰值浮点性能提高到原来的两倍：Opteron X4 核可以在每个时钟周期发射两条浮点 SSE2 指令，而 Opteron X2 核最多只能发射一条。由于我们比较的两个系统具有接近的时钟速率——Opteron X2 为 2.2GHz，Opteron X4 为 2.3GHz——所以 Opteron X4 的峰值浮点性能是 Opteron X2 的四倍，而两者的 DRAM 带宽完全相同。Opteron X4 还有一个 2MiB L3 cache，而 Opteron X2 没有。

图 6-19 比较了两个系统的 Roofline 模型。正如我们所期望的那样，脊点向右移动了，从 Opteron X2 中的 1 移动到 Opteron X4 中的 5。因此，为了看到下一代 Opteron 处理器性能的改进，核心程序的算术强度必须大于 1，或者核心程序的工作集必须适合 Opteron X4 的 cache。

Roofline 模型给出了性能的上限。假设你的程序远低于该上限。那么应该进行哪些优化呢？以什么顺序执行这些优化？

为了克服计算瓶颈，以下两种优化可以改进几乎任何核心程序：

1. 浮点运算组合。计算机的峰值浮点性能通常需要几乎同时的等量加法和乘法运算。这种均衡不仅是因为计算机支持融合的乘加指令（见 3.5.7 节的“详细阐述”），也因为浮点单元具有相同数量的浮点加法器和浮点乘法器。最佳性能也需要大部分浮点运算和非整数指令混合。

2. 提高指令级并行性并应用 SIMD。对于现代的体系结构，最高性能在每个时钟周期取指、执行并提交 3 ～ 4 条指令时才能获得（见 4.10 节）。这一步的目标是从编译器角度改进代码以增加 ILP。正如我们在 4.12 节看到的，一种方法是循环展开。对于 x86 体系结构，单个 AVX 指令可以操作 4 个双精度操作数，因此应尽可能使用它们（见 3.7 节和 3.8 节）。

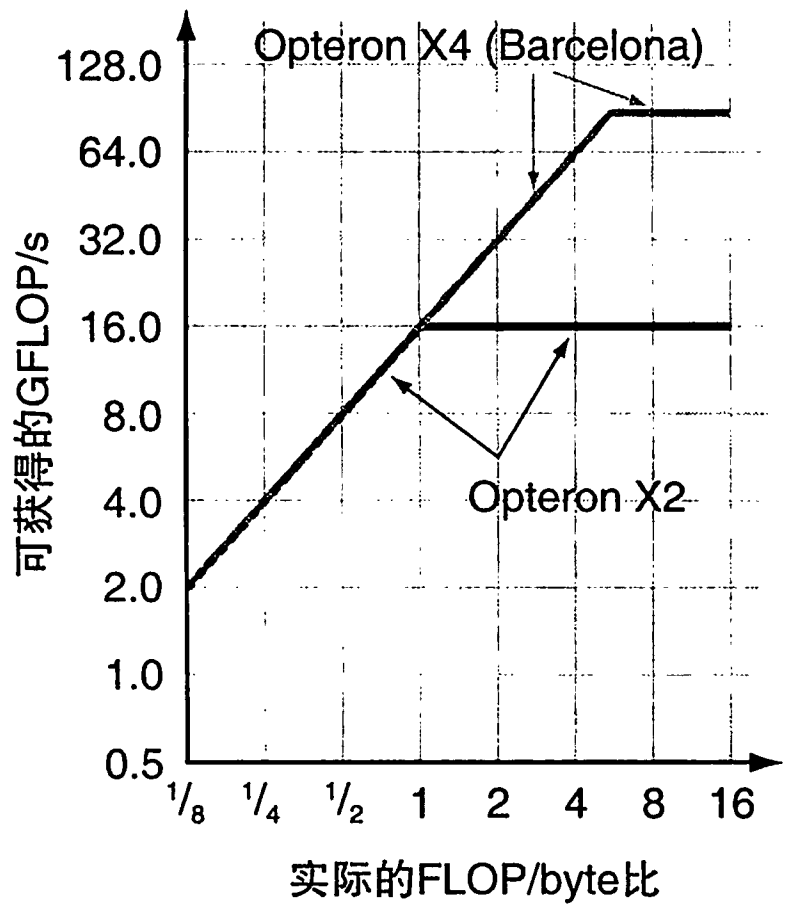


图 6-19　两代 Opteron 的 Roofline 模型。Opteron X2 的 Roofline 与 图 6-18 相同，使用黑色绘制，Opteron X4 使用灰色绘制。Opteron X4 较大的脊点意味着原来在 Opteron X2 中是计算受限的核心程序在 Opteron X4 中可能是存储性能受限

为了克服内存瓶颈，可以采用以下两种优化方法：

1. 软件预取（software prefetching）。通常，最高性能需要保持许多访存操作一直运行，通过执行软件预取指令来预测访存可以更加容易地满足这个需要，而不是等到计算需要该数据时才进行访存。

2. 内存关联（memory affinity）。现在大多数的微处理器都在片内包含了内存控制器，它能提高存储层次的性能。如果系统中含有多个芯片，这就会使一些地址访问本地 DRAM，而其他地址需要通过芯片互连才能访问对于其他芯片是本地的 DRAM。这种分裂会导致我们在 6.5 节介绍的非一致性访存。通过另一个芯片访存会降低性能。第二个优化是尽量将数据和操作该数据的线程分配给相同的存储器－处理器对，这样处理器很少需要访问其他芯片上的存储。

Roofline 模型可以帮助确定选择哪些优化方法，以及以什么顺序执行优化。我们可以认为这些优化的每一个都是适当的 Roofline 下方的“天花板”，也就是说，如果不执行相应的优化，就不能突破天花板。

计算性能 Roofline 可以在手册中找到，而存储 Roofline 则可以通过运行流式基准测试程序获得。计算性能的上限，例如浮点均衡，也可以来自该计算机的手册。内存性能的天花板，例如内存关联，需要在每台计算机上运行实验以确定它们之间的差距。好消息是，这个过程只需要在每台计算机上进行一次，只要有人完成了对该计算机天花板的评估，任何人都可以使用该结果指导该计算机优化的先后次序。

图 6-20 为图 6-18 中的 Roofline 模型添加了天花板，其中上图给出了计算天花板，下图给出了存储带宽天花板。尽管较高的天花板没有标记，但是其隐含使用了全部优化手段；为了突破最高的天花板，首先必须突破所有下面的天花板。

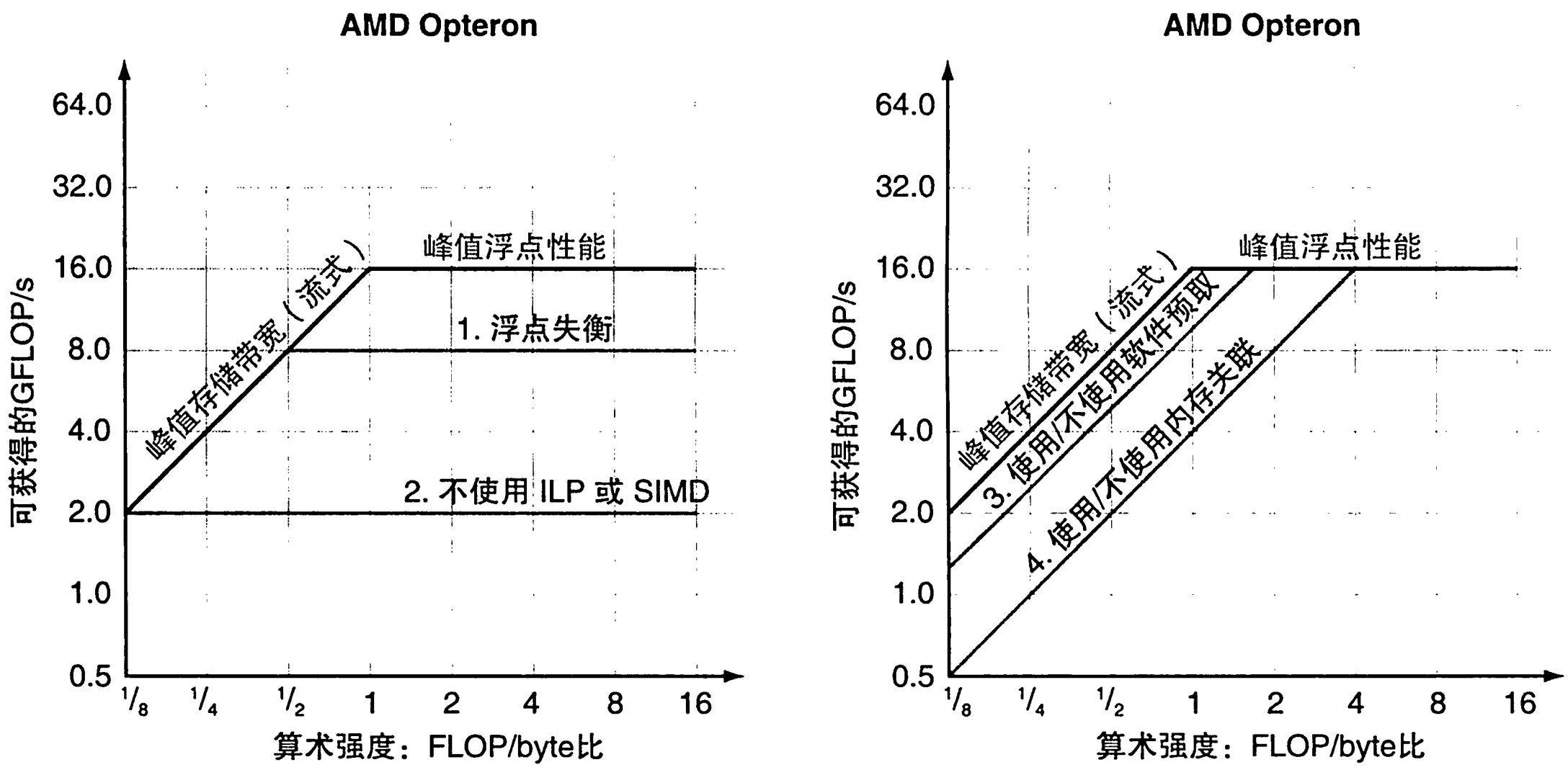


图 6-20　带天花板的 Roofline 模型。上图表示计算性能的“天花板”，1 表示浮点运算失衡情况下性能为 8GFLOP/s，2 表示同时未使用 ILP 和 SIMD 下的性能为 2GFLOP/s。下图表示没有软件预取的存储带宽上限为 11GB/s，如果同时没有内存关联优化，则为 4.8GB/s

天花板之间间隙的宽度和下一个更高的限制表示优化之后的收益。因此，图 6-20 建议优化 2 和 4。其中 2 是改善 ILP，对于改善该计算机的计算有很大益处；4 是改善内存关联，