第 4 章　工作负载驱动的性能评价

计算机体系结构领域变得越来越定量化，只有在对折中方案的详细评价完成后才能决定采用什么样的设计特性。系统一旦建立，理解折中方案的体系结构设计者和做出购买决定的用户都会对系统进行评价和比较。在单处理器设计中，有现存的机器及在其上面运行的应用这样丰富的基础，识别和评价折中方案的过程体现出从已知量出发的细致推断。设计者通过使用一些微基准测试程序——即一些强调机器特定性质的小程序，来分离机器的执行特征。一些流行的工作负载已经被编写在标准的基准测试程序集中了，比如，标准性能评价公司（Standard Performance Evaluation Corporation，SPEC）的工程计算类工作负载基准测试程序集（SPEC 1995），其测量是建立在一系列现存的不同设计方法之上的。基于测量、对新兴技术的评估以及应用需求的预期变化，设计者提出一些新的方法。对其中有前途的方法通过模拟来进行典型的评价。首先，要写一个模拟器。模拟器是一个程序，它模拟那些具有或者不具有所建议的特性的设计。其次，要选择一定数量的程序或者多道程序工作负载，它们可以是来自标准的基准测试程序集，或者是那些很可能要运行在该机器上的工作负载的代表。这些程序运行在模拟器上，机器特性对性能的影响也就决定了。特性的性能和实现该特性的硬件以及设计该特性的时间的预期成本一起，决定了是否要在机器中包括这种特性。模拟器要写得灵活，这样可以通过改变组成结构和性能方面的参数来理解它们的影响。

好的工作负载驱动的评价是一个困难而且耗时的过程，即使对于单处理器系统也是如此。当技术和使用模式发生变化时，工作负载需要更新。作为工业标准的基准测试程序集每隔几年就要修正一次。特别是，程序使用的输入数据集会影响到与系统的一些关键相互作用，并且决定是否能突出对系统的这些重要特性的评价。我们必须理解这些相互作用，并且反映到工作负载的使用中。比如，考虑到处理器速度的剧增和高速缓存尺寸的变化，从 SPEC92 到 SPEC95 的基准测试程序集的一个主要变化是使用了更大的输入数据集来加强存储器系统的测试。当然，精确的模拟器的开发和验证是很昂贵的，而且模拟程序的运行会消耗
199 大量的计算时间，但是，这些努力仍是值得的，因为一个好的评价会产生好的设计。

随着多处理器体系结构的成熟、从一代机器到下一代机器之间有着更多的连续性，人们采用类似的定量的评价方法。尽管早期的并行机设计在很多方面像艺术品的大胆创作，很大程度上依赖于设计者的直觉，但现代的设计却包含了对于所建议的设计特性的大量的评价。这里，工作负载既用来评价真正的机器，也用来推断所建议的设计是否合理，通过软件模拟来探索可能的折中方案。对于多处理器，所感兴趣的工作负载可能是并行程序，也可能是串行和并行程序混合的多道程序。对于多处理器体系结构，评价是新工程方法的一个关键部分。在考察多处理器体系结构的核心或者本书中所评价的折中方案之前，理解评价的关键问题是非常重要的。

不幸的是，对于多处理器体系结构的工作负载驱动的评价工作比在单处理器情况下要更加困难，其原因有以下几点：

- 并行应用的不成熟性。对于多处理器，找到有“代表性”的工作负载并非易事，这

既因为它们的使用相对来说不成熟，又因为有很多新的行为特征要表示。

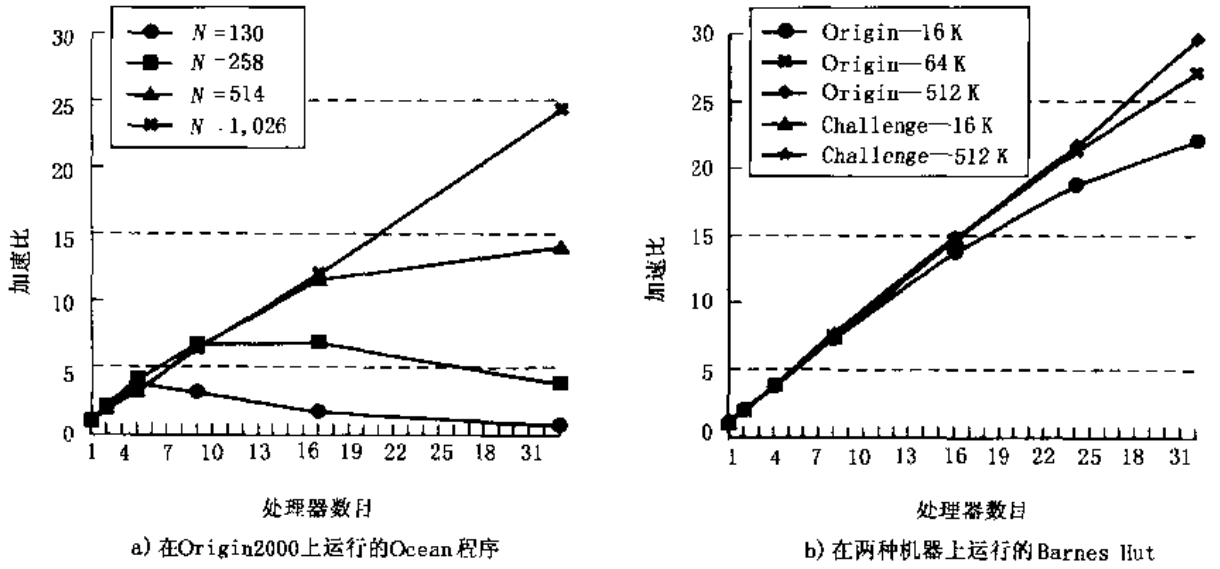
- 并行程序设计语言的不成熟性。并行程序设计的软件模型还不稳定。基于不同的模型编写的程序会有完全不同的行为。
- 行为差异的敏感性。不同的工作负载，甚至将相同的串行工作负载并行化时所做出的不同决策，都会呈现出差异巨大的体系结构执行特征。
- 新的自由度。在体系结构中有一些新的自由度。最明显的是处理器的数量。其他的自由度包括扩展的存储器层次结构，特别是通信体系结构的组成结构和性能参数。与工作负载的自由度（即应用程序参数）和基本的单处理器节点一起，这些参数导致了试验设计的极其巨大的空间，特别是在通用的环境中评价一个概念或者折中方案，而不是评价某一台确定的机器时更是如此。通信的高代价使得性能对所有这些自由度之间的相互作用比在单处理器情况下更为敏感。也使得对如何遍历大的参数空间的理解更加重要。
- 模拟的限制。用软件来模拟多处理器从而评价设计上的决策比模拟单处理器需要更多的资源。多处理器的模拟要消耗大量的内存和时间。因此，虽然我们希望探索的设计空间较大，但是，实际可能探索的空间往往要小得多，我们在决定对哪部分空间进行模拟时必须要做出细致的折中方案。

在应付这些困难时，我们对于第 2、3 章中的并行程序的理解是十分关键的。通过本章的学习，我们将了解到，要实现一个有效的模拟需要理解工作负载和体系结构两者的重要特 200
性以及这些特性是如何相互作用的。特别是，应用程序参数和处理器的数量之间的关系决定了程序的基本特性，比如通信与计算比、负载平衡、时间和空间的局部性等。这些特性和扩展的存储器层次结构的参数之间相互作用，以与应用相关的显著方式影响着程序的性能（见图 4-1）。选择工作负载和机器的参数值（或者规模），理解它们规模变化之间的关系，是工作负载驱动的评价的一个关键方面，有深远的意义。它会影响我们为了充分覆盖行为特征所设计的试验方案，也会影响我们评价的结论。它还能够帮助我们限制试验的次数或必须考察的参数组合数。

本章的一个重要目标是揭示这些特性和参数之间的关键相互作用，说明它们的重要性，并指出一些重要的误区。尽管对评价而言，没有普遍适用的规则，但本章力图详细阐述一种通过模拟来评价真实机器和评估折中方案的方法。在本章最后在对几种工作负载的特征化过程中，以及在贯穿全书使用这些工作负载的例证性的评价中，我们遵循了这个方法。重要的是，我们不仅要进行良好的评价，而且还要理解评价研究的局限性，这样，我们在做出体系结构决策时才能正确地使用评价这一工具。

本章开头讨论了随着处理器数量增加，放大工作负载参数的基本问题，研究了性能指标和并行程序关键的固有行为特征的含义。在接下来的两节里，讨论了与扩展存储器层次结构的组成结构和性能参数间的相互作用，以及如何将这些相互作用结合到实际的实验设计中去，这两节考察了两种主要类型的评价。

4.2 节概述了一种评价真实机器的方法，它包括：首先理解我们可能会使用到的基准测试程序工作负载的类型和它们在这种评价中的角色——包括微基准测试程序、内核、应用程序和多道程序式的工作负载，还要理解选择它们时的期望的标准。然后，对于给定的工作负载，我们考察在评价一台给定的机器时如何选择工作负载的参数，说明重要的考虑和可能的



a) 在Origin2000上运行的Ocean程序　　b) 在两种机器上运行的Barnes Hut

图 4-1　应用程序参数对并行性能的影响。对 Ocean 而言，所显示的应用程序的参数是每一维网格点的个数（N），而在 Barnes-Hut 中，它是星体的数量。这些参数决定了要使用的数据集的大小。对于许多应用程序而言，比如 a）中的 Ocean，参数的作用是很明显的，至少是当数据集的尺寸相对于处理器的数量是足够大之前。对于最小的问题，从 4 个到 8 个甚至更多处理器的情况下，性能变得更差而不是更好。对于仅次于最小的问题，当处理器个数从 8 到 16 时，性能下降。对于最大的问题，性能大致随处理器个数线性增长，直到处理器个数达到 32。对于其他的应用，比如 b）中的 Barnes-Hut，数据集尺寸的影响要小得多

误区。这一节的最后将讨论可能用于解释和表示结果的各种指标。4.3 节把这种方法的讨论扩展到更具有挑战性的问题，即在更一般的情况下，通过模拟来评价体系结构方面的折中方案。

在理解了如何执行工作负载驱动的评价之后，我们进入 4.4 节，它提供了工作负载的相关特征，在本书所介绍的例证性评价中要用到这些特征。在附录中，还列出了一些用于并行计算的重要的、可以公开获得的工作负载集以及它们的原理。

4.1　改变工作负载和机器的规模

在考察规模性能模型和它们的含义之前，我们首先讨论一下在多处理器上的一些基本的性能测量，从而可以体会到适当的规模扩放的重要性。

4.1.1　多处理器性能的基本测量

假设我们已经选择了一个并行程序作为工作负载，并想利用它来评价一台机器。对于一台并行机，我们可以测量它的两个性能特征：绝对性能和并行性所带来的性能改进。后者的测量通常称作加速比，它已在第 1 章中定义，等于 p 个处理器上的绝对性能除以在一个单处理器上的性能。绝对性能（和代价一起）对于最终用户或者说机器的购买者来讲是最重要的。但是，它本身并不能说明性能的改善在多大程度上是来自于并行性的应用和通信体系结构的有效性，而不是来自于底层的单个处理器节点的性能。加速比可以告诉我们性能有多少是来自并行性的使用，但是值得注意的是，当计算比较慢时，通信代价显得不是那么重要，所以当单个节点的性能比较低时，很容易获得好的加速比。这两种指标都很重要，都应该被

201
202

测量。

对绝对性能的最好的测量是单位时间内完成的工作量。给定一个程序，要完成的工作量通常由程序的操作所依赖的输入配置来决定，这被称为问题规模（我们以后将会更精确地定义问题规模）。输入配置可能只是在程序最前面使用，也可能包括要到达“服务器”应用程序的一系列连续的输入的集合，比如一个处理银行事务的系统或是对来自传感器的输入做出响应的系统。假设输入配置和由此引起的工作量对于一组实验来讲是保持不变的，那么我们就可以把工作量看作是一个固定的参考点，测量执行时间并且将性能定义为相应的执行时间的倒数。

用户发现，在一些应用领域中，即使当输入配置固定不变时，使用一个显式的、领域特定的工作量的表示和使用每单位时间所完成的工作量这样显式的性能指标更为方便。例如，在一个事务处理系统中，指标可以是每分钟服务的事务的次数；在排序应用中，指标可以是每秒钟排序的关键字的数量；而在化学应用中，指标可以是每秒钟计算的结合的数量。但是，尽管可以显示地表示工作量，性能还是相对于特定的输入配置或者工作量来测量的，这些性能指标仍然都是从执行时间（和涉及到的应用事件的数量一起）的测量中得来的。给定了一个固定的已知的问题配置，这些领域特定的指标并没有显示出优于执行时间或者执行时间倒数的根本优势。实际上，我们必须要慎重，确保所使用的显式的工作量的测量从应用程序的角度上来看确实是有意义的测量，而不是我们可以用来骗人的东西。随着讨论的深入，还会进一步地讨论工作指标的理想特性，在 4.2.5 节中会考虑到关于指标的更为细节的问题。现在，让我们先集中精力评价一下由于并行性带来的绝对性能的改善，即因为使用了 p 个处理器而不是一个处理器所获得的加速比。

把执行时间作为我们的性能指标，在第 1 章中，我们看到了可以简单地在一个处理器和 p 个处理器上运行具有相同输入配置的程序，性能改善或加速比的测量是：

$$\frac{\text{时间（1 个处理器）}}{\text{时间（}p\text{ 个处理器）}}$$

以每秒钟的操作次数作为性能指标，我们可以按下式测量加速比：

$$\frac{\text{每秒钟的操作（}p\text{ 个处理器）}}{\text{每秒钟的操作（1 个处理器）}}$$

203

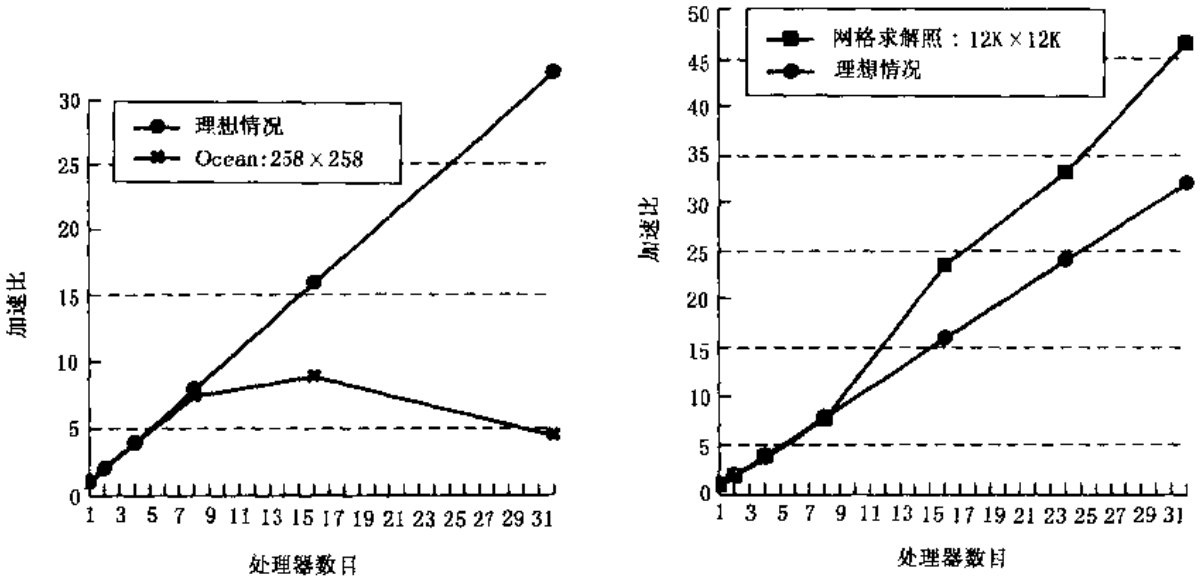
这里产生了一个问题，就是我们如何测量一个处理器的性能？例如，在一个处理器上运行最好的串行程序是否比在一个处理器上运行并行程序本身所得到的性能更为精确？这个问题其实是很容易回答的。随着处理器数量的变化，我们可以让这个问题运行在不同处理器数量的机器上，并且计算它们的加速比。那么为什么又要考虑扩放呢？

4.1.2　为什么要考虑扩放性

不幸的是，存在一些原因使得我们认为，把对固定问题规模的加速比测量作为评价各种不同规模的机器的并行性所带来的性能改善的惟一方法是不够的。

假设我们已经选定的固定问题规模相对较小，适用于只有几个处理器的机器。对于相同的问题规模，当我们增加了处理器的个数时，并行性所产生的额外开销（通信、负载失衡）相对于有效的计算就会增加。最终我们会到达某一点，问题规模对于评价现有的机器变得不

合情理的小。过高的额外开销会导致小得没有意义的加速比，其结果是由于使用了不合适的问题规模，因而不能反映出机器的能力（也就是说，问题对大型机器没有足够的并发性）。事实上，在有些点上使用较多的处理器甚至可能会损害性能，因为额外开销相对于有用的工作占据了主导的地位（参见图4-2a），用户不应该在那么大的机器上运行这类问题，所以用这样的问题来评价机器是不合适的。同样的情况，如果在大机器上运行耗时很少的问题也是不合适的。



a) 小的Ocean问题　　b) 大的方程求解器问题的超线性加速比

图4-2　当处理器个数增加时，SGI Origin2000的加速比。a）显示了在Ocean应用中小问题规模的加速比。该问题规模显然非常适合于有8个处理器的机器。在稍稍超过16个处理器时，加速比会达到饱和，此时我们不清楚是否应该在这么大规模的机器上运行这样规模的问题。为评价具有32个或者更多处理器的机器而运行这样规模的问题显然是不行的！b）显示了对方程求解器内核的加速，说明了超线性加速出现的条件，即在16个处理器的系统中，处理器的工作集能够容纳于高速缓存之中，而在使用8个或者更少的处理器时，处理器的工作集就无法被高速缓存所容纳

另一方面，如果我们选择了一个适用于运行在有很多处理器的机器上的问题，我们在评价由于并行性而引起的性能改进时可能会碰到相反的问题。这个问题对于单处理器来讲可能太大了，因为它的数据已经大到无法放入一个单节点的内存中。在某些机器中，它可能不在一个单处理器上运行；在另外一些机器中，单处理器的执行会使磁盘遭受严重的反复存取的颠簸；而在其他的一些机器中，溢出的数据会被分配到扩展存储器层次结构的其他节点的内存中，导致很多人为的节点间的通信。当使用足够多的处理器时，数据就会容纳于它们的集合的内存中；如果数据分布得合理，就会消除人为的通信。在每个处理器上的计算会更加有效，结果是加速比要大大高于所使用的处理器的数量。一旦发生这种情况，当处理器的数量增加时，加速比会以更为平常的方式得到进一步的改善，但是相对于单处理器的加速仍然和处理器的数量呈超线性。

这种情况可能出现在存储器层次结构的任何层次上，而不仅仅是发生于主存。比如说，当每一个具有自己高速缓存层次结构的处理器被加入时，机器的集合高速缓存容量会增加。如果每个处理器的工作集随着数据集减少，当处理器数目增加时，处理器开始更加有效地使

204
205

用它们的高速缓存。图 4-2b 说明了方程求解器内核使用高速缓存容量的例子。这种因为存储系统的效果而产生的巨大的超线性加速比不是假的。确实，从用户的观点看，具有更多的、分布式的内存是并行系统胜过单处理器工作站的一个重要的优点，因为它能够使并行系统运行更大的问题，而且运行得更快。但是，超线性加速比使我们无法区别容量带来的效应和并行化所带来的通常意义上的改善，也不能帮助我们评价机器通信体系结构的有效性。

当处理器个数增加时，保持不变的问题规模的最后限制是它也许不能反映机器实际的使用情况。用户经常想使用更加强大的机器来解决更大的问题，而不是让同样的问题解得更快。在这样的情况下，因为在机器的实际使用中问题规模与机器规模一起增长，所以在评价机器时，也应该扩放问题的规模。这样的扩放可以克服刚才讨论的由规模不匹配而引出的问题，但是却失去了针对相同问题比较机器配置的简单性。

关于如何使问题规模适应机器规模的变化而改变，我们需要良好定义的规模扩放模型，这样我们才可以根据这些模型来评价机器。不管扩放模型如何，性能的度量总是单位时间内所完成的工作量。但是，如果问题规模被扩放，所做的工作并不是保持不变的，我们不能简单地通过比较执行时间来决定加速比。工作量必须被表示和测量。问题是如何去做？此外，我们还想理解扩放模型是如何影响程序特征的，比如通信与计算比率、负载平衡、扩展的存储器层次结构中的数据局部性等。为了简单起见，我们集中看一下单一的并行应用程序，而不是多道程序的工作负载。首先，我们需要清楚地定义那些曾被非正式使用的术语：扩放机器和问题的规模。

*扩放机器*的意思是使机器更加强大（或者相反）。可以通过使机器的任何一个成分，如单个处理器、高速缓存、内存、通信体系结构或者输入输出系统的能力更加强大，更加复杂或者更加快速的办法达到这个目的。一般来说，*机器规模*是刻画单个节点的处理能力、存储器层次结构、通信和输入输出能力的一个向量。机器的扩放包括改变向量中的某一个或者多个项。因为我们感兴趣的是并行性，所以我们就把机器规模定义为处理器的数量，而且我们假设对于每个节点，它的本地高速缓存、存储器系统以及每个节点的通信能力在机器扩放时保持不变。扩展一台机器意味着添加多个相同的节点。比如说，把一台有着 p 个处理器，$p \times m$MB 内存的机器扩展 k 倍，产生的是一台拥有 $k \times p$ 个处理器和 $k \times p \times m$ MB 内存的机器。

*问题的规模*指的是一个特定问题实例或者输入配置，它通常由输入参数向量而不是单个参数 n（如，一个 $n \times n$ 的 Ocean 网格或者 n 个粒子的 Barnes-Hut）指定。比如，在程序 Ocean 中，问题规模由向量 $V = (n, \varepsilon, \Delta t, T)$ 刻画，n 是网格每一维的尺寸（它说明了我们在表示海洋时的空间分辨率），ε 是用来决定多网格方程求解器收敛的容差系数，Δt 指时间分辨率（即时间步之间的物理时间），T 是所执行的时间步的数量。在一个事务处理系统中，问题规模由使用的终端个数、终端用户产生业务的速率、事务处理的混合等决定。问题规模是一个决定程序所做的工作量的主要因素。

必须把问题规模和数据集尺寸区别开来。*数据集尺寸*是在一个单处理器上运行程序所需要的存储器的数量。这本身不同于程序的*内存使用*，程序的内存使用是包括复制在内的并行程序使用的内存的总和。数据集的尺寸通常依赖于少数几个程序参数。比如，在 Ocean 中，数据集的尺寸完全是由网格规模 n 决定的，但是指令条数和执行时间则是依赖于其他的问题规模参数。因此，尽管问题规模向量 V 决定了应用程序的很多重要参数，比如它的数据

集尺寸、它执行的指令条数和它的执行时间，它还是有别于这些特征中的任何一个。

4.1.3 扩放的关键问题

206 给出了这些定义后，要扩展一个问题使之运行在较大的机器上，还要解决两个问题：

1）问题扩放应该在什么约束之下？为了定义一个扩放模型，当机器扩放时必须保证一些特性不变。这些特性或许包括数据集尺寸、每个处理器的内存使用、执行时间、每秒钟执行的事务处理的数量、分配给每个处理器的粒子数量或者矩阵行数。

2）问题如何扩放？也就是说，如何改变问题规模向量 V 中的参数来满足选定的约束？

为了简化讨论，我们先假设问题规模由单个参数 n 决定，然后考察在这个假设下的扩放模型及其影响。稍后，在 4.1.6 节中，我们将会考察扩放工作负载参数时这些参数之间相互参照的更为微妙的问题。

4.1.4 扩放模型和加速比的测量

用作扩放约束的基础特性可以被分成两类：*面向用户的特性*和*面向资源的特性*。面向用户的特性的例子是在 Barnes-Hut 中分给每个处理器的星体的数量；在矩阵乘法程序中为每个处理器的矩阵行数，在事务处理中为每个处理器向系统发出的事务的数量以及每个处理器执行的输入输出操作的数量。面向资源的约束的例子是执行时间和每个处理器使用的存储器总量。由于在不同的约束下执行扩放时，给定数量的处理器要完成的工作量是不同的，因此每一个这样的约束都定义了一个不同的扩放模型。究竟是面向用户还是面向资源的约束更加合适则取决于应用领域，构造基准测试程序的一项关键性任务是保证扩放约束对于所涉及的领域是有意义的。

在执行评价时，面向用户的约束通常更容易被遵循（比如，随着处理器数量的变化简单线性地改变粒子的数量）。但是，大规模的程序经常是在严格的资源约束之下运行的，资源约束在跨不同应用领域时更有普遍性（时间就是时间，存储器就是存储器，不管程序是处理粒子还是矩阵），因此我们将使用资源约束来说明扩放模型的效果。让我们针对为了在一台规模大 k 倍的机器上运行而扩展应用规模时的约束，考察一下三个最流行的面向资源的模型：*问题约束的（PC）扩放*，*时间约束的（TC）扩放*和*存储器约束的（MC）扩放*。

在 PC 扩放中，问题规模是固定的，也就是说，它根本没有被扩放，尽管前面也讨论过一些关于固定问题规模所带来的问题。不管机器上处理器的个数有多少，都使用相同的输入配置。在 TC 扩放中，完成程序所需要的墙钟（wall-clock）执行时间是固定的。问题被扩放，使得在大机器上新的问题的执行时间和在小机器上旧问题的执行时间相等（Gustafson 1988）。
207 在 MC 扩放中，每个处理器使用的主存储器的数量是固定的。问题被扩放，使得新问题使用的主存正好是老问题的 k 倍（包括数据复制）。因此，如果老问题刚好能容纳于小机器的内存，那么新问题则应刚好能容纳于大机器的内存。

在某些领域里可能有一些更加适合的专门化的模型，比如，在商业在线事务处理基准测试程序中，事务处理委员会（TPC）规定了扩放的规则，即产生事务的用户终端数量和被访问的数据库的规模随被评价系统的“计算能力”成比例地扩放，其测量遵循指定的方式。这和 TC、MC 扩放模型一样，符合资源约束条件的扩放经常需要经过一些实验才能找到合适的输入，因为资源的使用并不一定是简单地随着输入参数而扩放。存储器的使用通常是可以预

见的，特别是如果没有在主存中复制的需要时更是如此，但是预见一个输入配置在 256 个处理器的机器上的执行时间和另外一个输入配置在 16 个处理器的机器上的执行时间相等却是十分困难的。我们将进一步研究 PC、TC 和 MC 的扩放模型，看看在这些模型下，“单位时间完成的工作量”，即加速比，被转换成了什么。

1．问题约束的扩放

在 PC 扩放中的假设是，用户使用较大机器的目的是为了更快地解决相同问题，这是一种很寻常的情况。比如，如果一个视频压缩算法每秒钟只处理一帧图像，我们使用并行性的目标可能不是在 1 s 内压缩一个更大的图像，而是每秒钟压缩 30 帧图像，从而可以对这种大小的帧进行实时压缩。另外一个例子是，如果一个 VLSI 的布线工具要用一周的时间来布通一个复杂的芯片的版图，我们更感兴趣的可能是如何使用并行性来减少布线时间，而不是为一个更大的芯片布线。因为在工作/时间的性能定义中，有用的工作保持不变，加速比指标的公式可以简单地表示成：

$$\text{加速比}_{PC}\ (p\ \text{个处理器}) = \frac{1\ \text{个处理器用的时间}}{p\ \text{个处理器用的时间}} \tag{4-1}$$

2．时间约束的扩放

这个模型假设用户有一定的时间可以用来等待程序的执行，他们想在这个固定的时间内解决尽可能大的问题。（想像一下那种愿意在计算中心购买 8 小时机时的用户或者那些愿意等待一个通宵完成程序的运行，但是需要在第二天早晨获得结果以便分析的用户）。尽管在 PC 扩放中问题规模固定不变，但执行时间是变化的，而在 TC 扩放中，问题规模是增加的，执行时间则保持不变。因为性能是工作量除以时间，而当系统扩放时时间保持不变，加速比可以由在固定的执行时间内完成的工作的增量来测定：

208

$$\text{加速比}_{TC}\ (p\ \text{个处理器}) = \frac{p\ \text{个处理器的工作}}{1\ \text{个处理器的工作}} \tag{4-2}$$

问题是如何测量工作，如果我们把它测量成在一个单处理器上问题配置的实际执行时间，那么我们可能不得不在机器的一个处理器上运行一个较大（扩展）规模的问题，这样才能获得分母[⊖]。不幸的是，这种情况可能行不通，或者要花很长的时间，或者根本不可能运行。

工作指标的理想特点是它应该容易测量，而且尽可能地与体系结构无关。它应该很容易用一个仅仅基于应用的解析表达式来建立模型，我们不应该进行额外的实验来测量问题被放大后的工作量，工作量的测量也应该与算法的串行时间复杂度成线性比例（见例 4.1）。

例 4.1　为什么线性扩放的特点对于工作指标很重要？

解答：如果我们希望理想的加速比（忽略存储器系统的人为效应）和处理器的数量成比例，那么线性扩放特点是很重要的。为了理解这一点，假设我们在一个矩阵乘法程序中把正方形矩阵的行数 n 作为工作的指标。让我们完全忽略掉存储器系统的相互作用，如果单处理器系统问题有 n_0 行，那么它的执行“时间”或者它需要执行的乘法操作的数量，和 n_0^3 成比例。因为问题是确定的，我们所能期望 p 个处理器在相同时间内的最好情况是执行 $n_0^3 \times p$

⊖ 原文是“分子”。——译者注

次操作，它对应（$n_0 \times \sqrt[3]{p}$）×（$n_0 \times \sqrt[3]{p}$）的矩阵。如果我们把矩阵行数作为工作的度量，那么根据式（4-2），即使在这种理想的情况下加速比是（$n_0 \times \sqrt[3]{p}$）/ n_0 或者说$\sqrt[3]{p}$，而不是 p。使用矩阵（n^2）中的点数作为工作的指标，从这点讲来，也是不合适的，因为它导致了时间约束的理想加速比为 $p^{2/3}$。但是，使用 n^3（乘法操作的次数）作为工作指标会得到理想的加速比 p，因为这种度量和矩阵乘法串行时间复杂度 $O(n^3)$ 成线性比例。■

理想的工作量度量不仅要满足这些特点，而且从用户角度看应是一个直觉参数。比如，在使用一种叫基数排序的方法对整数的关键字进行排序时，串行的复杂度随待排序的关键字的个数线性增加，所以我们使用关键字作为工作的度量。但是在实际应用中是很难发现这种度量的，特别是当多个应用的参数都在扩放，并且以不同方式影响执行时间的时候更是如此。我们在实践中应如何测量工作呢?

如果无法找到一个具有理想特性的单个直觉参数，我们可以试着去发现一种容易从直觉参数推导，而且随着串行复杂度线性扩放的度量。一种流行的实现矩阵因子分解的 LINPACK 基准测试程序就是这么做的。大家都知道基准测试程序应该使用 $2n^3/3$ 次浮点操作来因子分解一个 $n \times n$ 的矩阵。其余的操作要么和它成比例，要么完全不占主导地位。如在例 4.1 中的矩阵乘法，操作的次数可以很容易从输入矩阵的维数 n 计算得到，且显然满足线性扩放
209 的特性，所以在基准测试程序中，它被用作工作的度量。

真正的应用程序经常有多个参数要扩放、因此更加复杂。只要我们有一个良好定义的规则来同时扩放参数，我们就可以构造一个具有期望特性的解析形式的工作的度量。但是，这样的工作量计算可能不再是简单或者直观的，它们对评价者或者基准测试程序的提供者有很多要求。而且，在复杂的应用中解析形式的预测往往被简化（比如，它们通常是平均的情况或者它们没有反映出可能相当重要的“实现”的行为），因此所执行的指令或操作的实际增长率可能和预期的不同。

在这样的情况下，一种普遍适用的经验式的技术是运行串行程序，测量机器操作形式的工作量。如果能知道某一类型的高层次操作，比如粒子之间的相互作用，总是直接与串行复杂度成比例，那么，我们就可以计算出运行时执行的操作的数量。在更一般的情况下，我们可以尝试测量在一个单处理器上运行该问题所花费的时间，假设所有的存储器访问都是在高速缓存命中并且花相同的时间（比如，单个周期），因此消除了由存储器系统引起的人为因素。这种工作测量反映了在运行程序的时候哪些机器指令被实际执行了，同时避免了颠簸和超线性问题，我们把它称作具有完美存储器的执行时间（注意，它和在 3.4 节中介绍的串行性-有用时间十分相近）。很多计算机拥有某种系统实用程序，允许对计算的情况进行收集，获得具有完美存储器情况的执行时间。如果没有这样的功能，我们必须求助于对某些高层操作发生次数的测量。

一旦我们有了工作量的测量，我们可以用式（4-2）计算 TC 扩放下的加速比。但是，要决定能产生期望的执行时间，因而满足 TC 扩放的输入配置，可能需要经过反复的求精

3．存储器约束的扩放

这个模型基于的假设是用户想在不考虑运行时间的情况下，运行尽可能大的程序而不使机器的内存溢出。比如，对于一个星体物理学家来讲，重要的是运行一个像 Barnes-Hut 这样的 n 个星体的模拟，该模拟包含了机器可以接受的最大数量的星体以便提高星体对宇宙采

样的分辨率。MC 扩放导致经常使用一个叫做*扩放的加速比*的性能改善指标，它被定义成在单个处理器上运行一个较大的（扩展的）问题花费的时间与该问题在扩展的机器上运行时间的比率。对于卖方来讲，这个指标通常是有诱惑力的，因为这样的加速比一般会很高。实际上，它是对一个非常大的问题测量问题约束的加速比，往往具有低的通信与计算比和充分的并发性，并且受益于由存储器和高速缓存容量产生的超线性效应。不管怎样，扩展了的问题并不是我们在 MC 扩放模型下运行在单处理器上的问题，所以这不是一个合适的加速比指标。

210

与前面的模型不同，在 MC 扩放模型下，工作和执行时间都不固定。作为惯例，使用工作除以时间作为性能指标，我们可以把加速比定义为：

$$\begin{aligned}\text{加速比}_{MC}(p\text{ 个处理器}) &= \frac{p\text{ 个处理器的工作}}{p\text{ 个处理器的执行时间}} \times \frac{1\text{ 个处理器的执行时间}}{1\text{ 个处理器的工作}} \\ &= \frac{\text{工作的增量}}{\text{执行时间的增量}}\end{aligned} \tag{4-3}$$

如果执行时间的增加仅仅是因为工作量增加，而不是因为并行性的额外开销所致，即如果不存在 MC 扩放中一般很少发生的存储器系统人为效应，那么加速比将是 P，这正是我们所希望得到的。工作量的测量和我们在前面讨论过的 *TC* 扩放的测量方法一样。

因为在 MC 扩放下数据集的尺寸比在其他模型下增长的快，所以并行额外开销的增长相对缓慢，加速比通常更好（忽略容量型人为效应）。MC 扩放确实是很多用户所希望的使用并行机的方式。但是，对于很多类型的应用来说，MC 扩放会导致一个严重的问题：执行时间（对于并行执行）会长得令人无法容忍。这个问题能出现在任何一个工作随问题规模的增长比存储器使用的增长快得多的应用之中（见例 4.2）。

例 4.2　矩阵分解是串行工作的增长比存储器使用的增长要快得多的简单例子。请说明在这种应用中，MC 扩放如何导致并行执行时间迅速增长。

解答：在矩阵分解中，对于一个 $n \times n$ 的矩阵，虽然数据集的尺寸和存储器的使用按 $O(n^2)$增长，在一个单处理器上的执行时间却以 $O(n^3)$ 增长。假设一个 10 000 × 10 000 的矩阵要使用大约 800 MB 的存储器，可以在一个单处理器的机器上用一小时完成矩阵的因子分解，那么考虑一个包含 1 000 个处理器的扩展的机器，在这台机器上，在 MC 扩放模型下，我们可以因子分解一个 320 000 × 320 000 的矩阵，因为几乎不需要在主存中进行复制。但是，并行程序的执行时间会增加到将近 32 个小时（即使假设有完美的 1 000 倍的加速比）。■

在 3 个模型中，时间约束的扩放越来越被认为是最可行的一个，但是，没有一个模型可以宣称自己是最适于所有应用和所有用户的。不同的用户有不同的目标，在不同的约束下工作，不可能在任何情况下都非常严格地遵循一个给定的模型。但不管怎样，对于在机器扩放条件下分析扩放的性能而言，这 3 个模型都是有用的综合的工具。

4.1.5　扩放模型对方程求解器内核的影响

我们先考察一个简单的例子——第 2 章的方程求解器内核，看一看它如何和不同的扩放模型相互作用，以及这些模型是如何影响它与体系结构相关的行为特征的。对于一个 $n \times n$ 的网格来说，简单方程求解器需要的内存为 $O(n^2)$。计算复杂度是 $O(n^2)$ 乘以最终达到收敛的迭代次数，我们可以保守地假设迭代次数是 $O(n)$（使数值从网格的一侧边界流动到另

211

外一侧所需要的迭代次数）。这导致了$O(n^3)$的串行计算的复杂度。

假设在任何情况下，由于并行性的加速比等于处理器的个数p，考虑一下在3种扩放模型下的执行时间和存储器需求。对于PC扩放，因为相同的$n \times n$网格在更多的处理器p之间分割，每个处理器的存储器需求会按p线性下降，执行时间也是一样。在TC扩放中，执行时间根据定义是固定不变的　假设是线性加速比，这就意味着如果扩展的网格规模是$k \times k$，那么$k^3/p = n^3$，所以$k = n \times \sqrt[3]{p}$。因此每个处理器需要的内存为

$$\frac{k^2}{p} = \frac{n^2}{\sqrt[3]{p}}$$

它按照处理器的立方根减少。根据定义，使用MC扩放，每个处理器的存储器需求保持不变，为$O(n^2)$，这里单个处理器执行的基本网格是$n \times n$。这意味着网格的整个规模增长了p倍，所以扩展的网格成为$n\sqrt{p} \times n\sqrt{p}$而不是$n \times n$。因为它现在需要$n\sqrt{p}$次迭代才能收敛，所以串行的时间复杂度为$O((n\sqrt{p})^3)$。这意味着即使假设由于并行化产生的加速比是理想的，在p个处理器上扩展了的问题的执行时间是

$$O\left(\frac{(n\sqrt{p})^3}{p}\right)$$

或者说是$n^3\sqrt{p}$。因此，基本问题的并行执行时间是串行执行时间的$\sqrt{p}$倍。甚至在线性加速比的假设下，在一个处理器上需要运行一个小时的问题，在*MC*扩放模型下在一台具有1 024个处理器的机器上要运行32个小时。那么，对于这个简单的方程求解器，MC扩放下的执行时间会快速增加，在TC扩放下每个处理器对内存的需求减少。

让我们考虑一下不同的扩放模型对于并发性、通信与计算的比率、同步和I/O频度、时间和空间的局部性、消息的尺寸（在使用消息传递模型时）的效应。

这个内核中的并发性和网格点的数量成正比。它在PC扩放下保持不变；在MC扩放下的增长和p成正比；在*TC*扩放下的增长和$p^{0.67}$成正比。

通信与计算的比率是分配给每个处理器的网格分区的周长与面积之比，也就是说，它和每个处理器的点数（n^2/p）的平方根成反比。在PC扩放模型下，该比率按$\sqrt{p}$增长；在MC扩放模型下，分区大小不变，所以通信与计算的比率也不变。最后，在TC扩放模型下，因为一个处理器的分区的规模随着处理器数量的立方根减少，则该比率按p的六次方根增加。

方程求解器在每个网格扫描的结尾处同步一次来决定收敛与否。假设它在那个时刻也执行I/O操作，比如，在每次扫描的结尾输出最大误差。在PC扩放模型下，在一次给定的扫描中每个处理器完成的工作随着处理器数量的增加而线性递减，所以假设线性加速比，同步频率和输入输出频率随着p而线性增加。在MC扩放模型下，频率保持不变；在TC扩放模型下，它随着p的立方根增加。

212

这个方程求解器的重要工作集的尺寸正好是网格在一个处理器中的分区的大小，表示了它的时间局部性。因此，在PC扩放下，重要工作集的尺寸和对高速缓存的需求随着p而线性减少，在MC扩放中保持不变，在TC扩放中随着p的立方根减少。因此，在TC扩放中，尽管总的问题规模在增加，每个处理器的工作集的尺寸还是在减小。

方程求解器的空间局部性在一个处理器分区内部，在面向行的边界上是最好的，在面向

列的边界上是最差的。因此，它随着处理器的分区变小，随着面向列的边界相对于分区面积变大而减小，因此在MC扩放下它保持不变，在PC扩放下减少得很快，在TC扩放下减少得比较慢。

最后，在消息传递模型中，一个单个的消息很可能包含一个处理器分区的边界行或边界列，它是分区尺寸的平方根。因此，这里消息尺寸的扩放类似于通信与计算的比率的扩放。然而，一个进程发送的消息数量只是依赖于相邻的进程数目，而与 n、p 或者扩放模型无关。

从前面的讨论中我们可以很清楚地看到，只要存储器或者高速缓存的容量效应不是主要的，在MC扩放下，我们可以期望有最低的并行性额外开销和最高的加速比，在TC扩放下其次。在PC扩放下，我们应该能够预料到，至少一旦额外开销相对于有用的工作变得显著起来时，加速比会很快下降。还有一点很清楚，应用参数的选择和扩放模型在很大程度上影响基本的程序特性以及和扩展存储器层次结构在体系结构上的相互作用，比如空间和时间的局部性。除非已经知道一个特定的扩放模型刚好适合一个应用程序或者特别不适合，以所有这3种模型评价机器都是有用处的。在讨论实际的评价时，我们会更仔细地考察与体系结构参数的相互作用以及它们对评价的重要性（4.2节和4.3节）。首先，我们先简要地看一看扩放的其他那些重要而精妙的方面：如何扩放应用的参数来满足给定的扩放模型的约束。

4.1.6 扩放工作负载参数

在讨论问题规模扩放的约束时，我们仅对单个应用参数 n 作了简化性的假设，但没有考察构成问题的规模向量的不同应用参数应该如何相互参照进行扩放来满足选定的约束条 213
件。我们现在不考虑这个简单假设，比如，Ocean应用程序有一个带4个参数的向量：n，ϵ，Δt，T。工作负载应该如何相互参照而扩放，这在PC扩放中不是问题；但是，它在MC、TC扩放中的确是一个问题。不同的参数彼此相关，不太可能扩放其中一个参数而不影响其余的参数，或者说不可能独立地扩放它们。比如，在Barnes-Hut应用的实际使用中，参数 θ（力的计算精度），Δt（时间步之间的物理时间间隔）应该随 n（星体的数量）的变化而扩放。所有这些参数决定了一个给定的执行特性，比如，Barnes-Hut的执行时间的增长并不是简单的 $n\log n$，而是

$$\frac{1}{\theta^2 \Delta t} n \log n$$

结果，在TC扩放模型下，星体数目 n 的增加没有仅仅由扩展 n 获得的那么大。

即使简单的方程求解器内核也有另外一个参数，ϵ，它是用来决定求解器收敛的容差系数。使该容差系数变小，正如实际应用中随着 n 的扩放所做的那样，就要增加为达到收敛所需要的迭代次数，因此，也就提高了执行时间；但是，它并没有影响对存储器的需求。和仅仅扩放 n 相比较，ϵ 和 n 一起扩展导致了在TC扩放下每个进程的网格尺寸、存储器需求、工作集尺寸减少得更快，通信与计算的比率在TC扩放中增加得更快，而在MC扩放中保持不变，甚至在MC扩放中执行时间增加的更快。作为一个使用工作负载的机器设计者，从应用程序用户的观点来理解参数之间的关系，并根据这种理解在评价中对参数进行扩放是十分重要的。否则，我们很容易获得一个不正确的关于体系结构的结论。

参数之间的实际关系和扩放它们的规则依赖于应用的领域。因为没有一个普遍适用的规则，这使得好的评价变得更加令人感兴趣。比如，在像 Barnes-Hut 和 Ocean 这样的通过使物理现象离散化而进行模拟的应用中，不同的应用参数通常精确地决定着模拟某些现象（比如星系的演化）过程中不同的误差来源。因此，整体扩放这些参数的合理规则是由扩放不同类型误差的策略所决定的。理想的情况是，基准测试程序集会描述扩放规则，甚至可以在应用中对它们编码，只留下一个像 n 这样的自由参数；所以对于作为基准测试程序使用者的设计者来说，他不必再操心学习这些东西。习题 4.12 和 4.13 说明了合适的应用程序扩放的重要性，它们显示了适当地扩放多个参数与仅仅扩放数据集尺寸参数 n 相比，往往会得出关
214 于体系结构的定量的、某些时候甚至是定性的不同结论。

4.2 评价一台实际的机器

现在，我们已经理解了合适扩放的重要性以及问题和机器的规模对于基本行为特性和体系结构相互作用的效果，我们可以为两种主要类型的工作负载驱动的评价制定特定的指导方针，这两类评价是：评价一台实际的机器和在通用的环境中评价一个体系结构的概念或折中方案。评价一台实际的机器在很多方面是比较简单的：组成结构、粒度和机器的性能参数是固定的，我们所要考虑的是选择合适的工作负载和工作负载参数，而且我们并没有受到软件模拟的局限性的约束。本节提供了评价实际机器的一个范例性模板。我们从使用微基准测试程序分离性能特征入手，然后考虑为评价而选择工作负载所涉及的主要问题。紧接着介绍一旦选定了工作负载后评价机器的指导方针，首先是何时固定处理器的数量，然后是何时允许它变化。本节最后将讨论用于测量机器性能和表示评价结果的流行的指标。在评价一台实际的机器时的所有问题都和评价体系结构的概念或折中方案有关。

4.2.1 使用微基准测试程序分离性能

评价一台实际的机器的第一步是理解它的基本性能指标，也就是，程序设计模型提供的主要操作的性能特性、通信抽象（用户/系统接口）或者硬件/软件接口。这通常是通过使用小的、专门编写的被称作*微基准测试程序*（Saavedra, Gaines, and Carlton 1993）的程序来完成的，微基准测试程序是为了分离这些性能特征（比如，延迟、带宽、额外开销等）而设计的。

在并行系统中使用了 5 种类型的微基准测试程序；前三个也可以用于单处理器的评价：

处理类微基准测试程序。测量处理器不涉及存储器访问的操作性能，比如算术操作、逻辑操作和转移。

本地存储器微基准测试程序。决定了本地节点内部的存储器层次结构各层的组成、时延、带宽，测量由不同层次满足的本地读和写操作的性能，包括那些导致 TLB 扑空和缺页的操作。

215 *输入－输出微基准测试程序*。测量 I/O 操作的特征，比如不同跨距和长度的磁盘读写。

通信微基准测试程序。测量数据通信操作，比如消息发送和接受或者不同类型的远程读写。

同步微基准测试程序。测量不同类型的同步操作的性能，比如加锁和栅障。

通信和同步微基准测试程序依赖于所使用的通信抽象或者程序设计模型。它们可能涉及

一个或者一对处理器，比如，一个单个的远程读扑空、一对发送/接收或者一个自由锁的获得；它们也可以是集合式的，比如广播、归约、全部到全部的通信、概率性通信模式、很多处理器竞争一个锁或者路障。可以设计不同的微基准测试程序来强调非竞争时延、带宽、额外开销和竞争。

出于测量的目的，微基准测试程序通常是采用重复的基本操作集合来实现的（比如，一行中有 10 000 个远程读操作）。它们通常具有一些简单的参数，这些参数可以变化以获得更加完整的特征；比如，集合通信微基准测试程序中涉及的处理器的数量，或者本地存储器微基准测试程序中连续读操作之间的跨距。图 4-3 说明了使用本地存储器微基准测试程序所获得的机器的典型情况。微基准测试程序的主要作用是分离和理解基本系统能力的性能。一个现在尚未实现的更具雄心的想法是，如果工作负载可以用不同的基本操作的加权之和来刻画，那么对于给定的负载一台的机器的性能可以根据它执行相应的微基准测试程序的性能做出预测。当我们在后续的章节中测量真实的系统时，我们将讨论一些特殊的微基准测试程序和它们的设计问题。

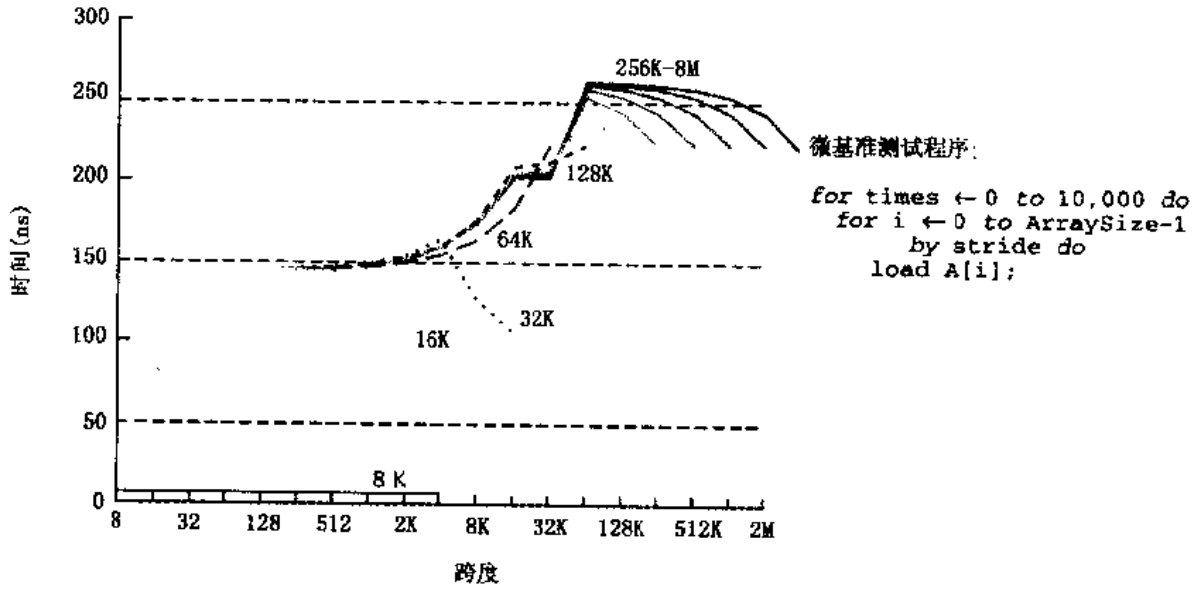


图 4-3 在 CRAY T3D 多处理器的单个处理节点上微基准测试程序的实验结果。每个处理器有一个由本地存储器支持的小的单级高速缓存。微基准测试程序由读取本地数组的大量操作构成，y 轴显示了每次读操作需要的时间，以 ns 为单位。x 轴是在一次循环中连续读操作之间的跨距（即所访问的存储器单元的地址的差异）。不同的曲线对应所访问的数组的尺寸（ArraySize），并标注了数组的尺寸。当 ArraySize 小于 8 KB 时，数组可以容纳于处理器的高速缓存，这样所有的读操作都能命中，需要 6.67 ns 完成读操作。对于大一些的数组，我们能观察到高速缓存扑空的效应。平均访问时间是命中时间和扑空时间的加权和，直到跨距大于缓存块的大小（32 个字或 128 个字节）而造成地址出界，使得每一次访问都扑空为止。曲线下一个上升是由于某些访问引起缺页而发生的，地址出界的原因是跨距足够大（16 KB），以至于每一次连续的访问都导致缺页。曲线最后一个上升是由于 4 体的主存中存储体的访问发生了冲突所致，当跨距为 64 KB 时，连续的访问总是落在相同的存储体上，而其余的存储体总是空闲的

分离了性能特征之后，下一步骤是使用更加实际的工作负载来评价机器。我们必须沿着三个主要的坐标：工作负载、它们的问题规模、处理器的数目（或者机器规模）进行遍历。

低级机器参数是固定的，我们首先来为评价选择工作负载。

4.2.2 选择工作负载

除了微基准测试程序以外，用于评价的工作负载可以按照真实性和复杂度上升的次序划分成三类：内核、完整的应用程序和多道程序的工作负载。每一类都有它自己的作用、优点和缺点。

内核是实际的应用程序中良好定义的部分，但是它们本身却不是完整的应用程序。其范围从简单内核（比如，一次矩阵转置或者一次相邻网格的扫描）到更为复杂的、占应用程序执行时间的主要部分的那些基本内核（比如矩阵因子分解和解偏微分方程用的迭代方法）。用于信息处理的内核的例子包括在决策支持应用中使用的复杂的数据库查询或者一组数据的排序。内核揭示了微基准测试程序中所没有的高层次的相互作用，结果，在某种程度上也失去了性能分离的作用。它们的关键特性是：它们那些与性能相关的特征，比如通信与计算比率、并行性和工作集，很容易理解并经常可以解析地决定，所以作为相互作用的结果观察到的性能可以根据这些特征来解释。

完整的应用程序由多个内核组成，并且展示了那些一个内核所不能展示的内核之间的高层次交互。完整的应用程序不同于内核，它们由用户运行以获得用户关心的答案。相同的大数据结构可以在一个应用程序中被多个内核以不同方式访问，而被不同的内核访问的不同的数据结构可能在存储器层次结构中产生相互的干涉。此外，对于孤立的内核最佳的数据结构在完整的应用程序中可能不是最好的。对于划分技术也是如此。比如，如果在一个应用程序中有两个独立内核，那么我们可以决定不把每个内核划分给所有的进程，而宁愿在它们之间共享进程。共享一个数据结构的不同内核可以有几种划分的方法，以求在它们不同的访问类型和通信模式之间得到某种平衡，产生最大的总体局部性。一个应用程序中多内核的存在带来了很多微妙的相互作用，一个完整的应用与性能相关的特性通常不能用解析的方法精确地决定。

216
?
217

多道程序工作负载是由多个在机器上一起运行的串行应用程序和并行应用程序构成的。不同的应用程序可以在时间上或者在空间上共享机器（比如，运行在机器处理器的不相交子集上的不同应用程序）或者同时具有时间共享和空间共享，它依赖于操作系统多道程序的规定。正如完整的应用程序因作为其组成成分的内核之间的高层次相互作用而变得复杂，多道程序的工作负载也涉及到多个完整应用程序本身之间的复杂相互作用。

随着我们从内核转移到完整的应用程序和多道程序工作负载，我们获得了真实性，这是非常重要的。很多致命的错误和性能问题不能被微基准测试程序甚至内核所揭示，但是却能被这些工作负载所发现。但是我们也失去了某些能力，包括：简明地描述工作负载的能力，明确地说明和解释结果的能力，孤立各种性能因素的能力。在极端情况下，多道程序工作负载不仅难以解释，而且设计都非常困难：比如哪些应用程序应该被包括在这种工作负载中？比例是什么？因为和操作系统之间有着精细的与定时相关的交互，从多道程序工作负载获得可重复的结果也是很困难的。每种类型的工作负载都有它们的作用，但是只能由完整的应用程序和多道程序工作负载反映出较高层次的相互作用这一事实（以及它们是由用户实际运行在机器上的工作负载的事实），这使得我们使用它们最终决定机器的整体性能变得十分重要。

让我们考察一下为评价而选择这样的工作负载（应用程序、多道程序负载、甚至是复杂

的内核）时期望的特性。这些特性包括应用领域的代表性、行为特性的覆盖性和足够的并发性。

1．应用领域的代表性

如果我们作为要购买机器的用户正在进行评价，而且我们知道机器会仅仅被用来运行某些类型的应用程序的话，那么，选择一个有代表性的工作负载就是一件容易的事情。另一方面，如果我们用机器运行范围宽广的工作负载，或者如果我们是试图通过评价一台通用的机器从而为下一代机器的设计得到一些启示的设计者，我们应该选择能代表宽广领域的工作负载的混合体。

今天，并行计算的某些重要领域包括：模拟物理现象的科学应用；像计算机辅助设计、数字信号处理、汽车碰撞模拟、甚至是用于评价体系结构折中方案的模拟这样的工程应用；
218
渲染场景或实体使之成为图像的图形化和可视化应用；像图像、视频和音频的分析和处理、语音和手写体文字识别这样的媒体处理应用；像数据库，数据挖掘和事务处理这样的信息管理应用；像航空公司员工调度和交通控制这样的优化应用；像专家系统和机器人这样的人工智能应用；多道程序工作负载；以及本身就是一个复杂的并行应用的多处理器操作系统。

2．行为特性的覆盖性

工作负载可以在第3章所讨论的与性能相关的特征的整个范围之内发生实质性的变化。其结果是，评价存在的一个主要问题是，它很容易利用工作负载进行欺骗或被工作负载所误导。比如，一项研究可以选择那些强调体现了体系结构优点（比如，通信时延）的特性的工作负载，避免使用那些性能比较差的工作负载（比如，本地访问、竞争或者通信带宽）。对于通用的评价，重要的是使我们选择的工作负载在整体上能强调重要的性能特征的范围。比如，我们应该选择能覆盖低的和高的通信与计算比、小的和大的工作集、规则和不规则的访问模式和本地的、远距离的或者集合式通信的工作负载。如果我们对评价特殊的体系结构特征（比如，处理器之间的全部对全部通信的总带宽）特别感兴趣，那么我们至少应该选择一些强调这些特征的工作负载。

另外一个重要的问题是程序优化的级别。真实的并行程序并不总是像第3章讨论的那样为获得好的性能而进行了高度优化，第3章所讨论的优化不仅仅是针对现有的特定机器，而是甚至采用了降低通信与计算比率或者提高时间和空间局部性这样更加一般性的方法。没有高度优化的原因或许是因为优化程序所涉及的工作量比用户愿意付出的多，或许是因为程序是在自动并行工具的帮助下生成的。优化的级别很大程度影响着关键执行特征和突出体系结构能力的程度。我们应该特别注意4种重要类型的优化：

算法。任务的分解和分配可能不够最优（比如，比较一次网格计算是采用面向条的分配还是面向块的分配（见2.3.3节））；对数据局部性的算法上的增强（比如，阻塞）可能并没有被实现。

数据结构。使用的数据结构可能不是最优地与体系结构相互作用，增加了人为的通信，比如，比较在一个共享地址空间中是用二维数组还是四维数组来表示一个二维的网格（见3.3.1节）。
219

数据的规划、分布和对齐。即使使用了合适的数据结构，它们也可能没有被适当地分布或者没有合适地对齐页或者高速缓存的块，这会导致过高的本地流量或者人为通信。

通信和同步的协调。所产生的通信和同步结构可能不够优化，比如，在消息传递型系统

中发送小消息而不是大消息。

因为优化经常是自组织的，这些分类强制了某些结构。在合适的地方，我们应该将机器或特性的健壮性与具有不同级别优化的工作负载相比较。

3. 并发性

在工作负载中主要的性能瓶颈可能是计算负载的不平衡性，究其原因或者是分割方法所固有的特性、或者是协调同步的方法（比如，使用路障而不是点对点的同步）所致。如果这是事实，那么该工作负载可能对于评价该机器的通信体系结构不合适，因为体系结构对于这种瓶颈是无能为力的。甚至通信性能的巨大改进也不可能对整体性能有太大影响。为了评价通信体系结构，我们应该保证我们的工作负载和它们的问题规模呈现足够的并发性和负荷平衡。这里一个有用的概念是*算法加速比*，该加速比是假设所有的存储器访问和通信操作不花费时间（见第3章中 PRAM 体系结构模型的讨论）。通过完全忽略数据访问和通信的性能影响，算法加速比测量了工作负载的计算负载平衡性以及并行程序额外要做的工作。

一般来说，我们应该分离由于工作负载特征所引起的性能限制，对这些限制机器本身是无能为力的。另一件重要的事是，工作负载应该运行足够长的时间，使得它对于所评价的机器规模而言是真实的负载，尽管这一点和并发性通常更应该是输入问题规模的函数，而不是工作负载本身的函数。

连同前面讨论过的那些标准，人们已经为定义并行应用的标准基准测试程序集为方便工作负荷驱动的体系结构评价付出了巨大的努力。基准测试程序集覆盖了不同的应用领域具有不同的原则，其中一些在附录中将予以描述。尽管用于本书中例证性评价的工作负荷集是非常有限的，它们却是根据前述标准而选择的。现在，让我们假设一个特定的并行程序已经被选作工作负载，我们要看一看如何利用它来评价一台真实的机器。首先，我们使处理器的数量固定，这既简化了讨论，也能更加清楚的显现出关键的相互作用。然后，再改变处理器的
220 数量。

4.2.3 评价一台固定规模的机器

固定了工作负载和机器的规模，我们只需要选择工作负载的参数。我们已经看到，对于固定数量的处理器改变问题规模会显著地影响所有重要的执行特征，从而影响评价的结果。实际上，它甚至会改变主要瓶颈的性质，即主要瓶颈是通信、负载失衡或本地数据访问。这告诉了我们非常重要但是经常被忽视的一点：在评价中仅仅使用惟一的问题规模往往是不够的，即使当处理器的数量固定时也是如此。

我们可以利用对于应用程序和体系结构的相互作用的理解来选择待研究问题的规模。我们的目标是对于真实固有行为以及体系结构方面的相互作用有足够的覆盖性，而同时又限制所需要的不同问题规模的数量。我们用一系列有计划的步骤来实现这一目标并显示在这一过程中只选择单一规模是错误的做法。我们的讨论会每次前进一步，在每一个步骤中，我们使用简单的方程求解器内核来定量地说明问题。为了得到定量的说明，我们假设正在评价一台有 64 个单处理器节点，每个节点有 1 MB 高速缓存和 64 MB 的主存，具有高速缓存一致性的共享地址空间的机器其步骤如下：

第 1 步：确定问题规模的范围

适用于某些幸运情况的一种选择问题规模的方法是要求更强的能力。我们进行的研究的

高层次目标有助于我们选择问题的规模。比如，我们可能知道机器的用户只对一些特定的问题规模感兴趣，这就简化了我们的任务。但是这种情况不常见，而且不是一个通用的方法。它不适用于方程求解器的内核。

对实际使用的知识可以帮助我们确定一个范围，低于此范围的问题对机器来说规模小得不实际，高于此范围的问题对于机器来说执行时间又太长或者用户不感兴趣。这对方程求解器的内核也不是特别有用，一旦我们确定了范围，就可以开始下一步了。

第 2 步：使用固有的行为特征

固有行为特征（比如通信与计算比和负载平衡）能帮助我们进一步限制范围并在已经选定的范围中选择问题规模。因为固有的通信与计算比常常随数据集规模的上升而降低，所以大问题可能没能足够地强调通信体系结构，至少对于固有的通信是如此；而小问题对通信的强调可能不具代表性，而且可能掩盖其他的瓶颈。因为并发性通常随着数据集规模的增加而增加，我们可以至少选择一些足够大但又不至于大到使得固有的通信变的太小的问题规模来 221
达到负载平衡（见例 4.3）。问题规模也可以影响到应用在不同阶段所花费的执行时间，不同的阶段可能会有完全不同的负载平衡、同步和通信的特征。比如，在 Barnes-Hut 应用的实例研究中，较小的问题在树形成阶段花费了更多的时间，它不能很好地并行，与在实际中通常占主导地位的引力计算的阶段相比缺少一些期望的特性。因此我们应该小心不要选择没有代表性的场景。

例 4.3　你如何利用固有行为特性为方程求解器的内核选择问题规模的范围？

解答：对于这种内核，足够的工作量和负载平衡可能要求我们至少有 32 × 32 个点的分区，对于一台 64 个（8 × 8）处理器的机器，这意味着整个网格的规模至少是 256 × 256。这个网格规模对于 32 × 32 或者 1 K 个点的计算，需要在每次迭代中每个进程有 4 × 32 或者 128 个网格点的通信。如每个网格点有 5 次浮点运算和 8 个字节，固有的通信与计算比是每 5 次浮点运算 1 个字节。假设一个处理器能为该计算提供 200 MFLOPS（每秒 2 亿次浮点运算），这意味着需要 40MBps 的带宽。这对现代的多处理器网络来讲是相当小的，即使它是突发形式的通信也不算什么。我们假设低于 5 MBps 的通信对我们的系统来说很小，仅仅从固有特性的观点出发，不需要在每个处理器上运行大于 256 × 256 点（64 KB × 8 或者 512 KB 的数据）的问题，或者说不需要整体上大于 2 K × 2 K 的网格。■

像负载平衡和通信这样的固有特征随着问题规模而平滑地变化，所以要单独处理它们。可以选择少数几个跨越感兴趣范围的问题规模，如果它们的变化速率非常低，可能不需要选择很多种规模。实验显示，在大多数情况下 3 个左右是刚好的。比如，对于方程求解器的内核，我们可以选择 256 × 256，1 K × 1 K 和 2 K × 2 K 的网格。

另一方面，和体系结构发生相互作用的时间和空间局部性展示了它们在性能效果上的阈值，包括当问题规模变化时人为通信的产生。我们可能需要扩展对于问题规模的选择来获得对这些阈值的足够的覆盖。同时，阈值性质能帮助我们剪裁参数空间。选择问题规模的下一步骤是考察时间局部性和工作集。

第 3 步：使用时间局部性和工作集

工作集是否能被本地高速缓存或者复制存储器所容纳会显著地影响执行特征，比如本地存储器流量和人为的通信，即使在固有的通信、计算负载平衡变化不大时也是如此。在像 Raytrace（光线跟踪）这样的应用中，重要工作集都很大，而且主要是由分配到远程节点的 222

数据构成，所以由于复制容量有限所引起的人为通信可能在固有通信中占主导地位。这种人为通信随着问题规模的提高在增加，而不是减少。在其他应用中（比如 Ocean），不能容纳于高速缓存的工作集会显著地产生更多的本地存储器流量而不是人为的通信。如果在实际中这样的问题规模是现实的话，我们应该包括代表了重要工作集阈值的两侧（即能够或不能够被高速缓存所容纳）的问题规模。实际上，只要对应用程序来说是现实的，我们也应该包括对于机器而言非常大的问题规模，比如，一个几乎填满整个内存的问题规模，尽管从负荷平衡和同有通信的观点来看它可能是没有多大意义的。大的问题经常导致体系结构和操作系统之间的相互作用，比如 TLB 扑空、缺页以及由于缓存容量型扑空而引起的巨大流量，而较小的问题则不会产生这样的效果。例 4.4 和例 4.5 有助于说明如何基于工作集选择问题的规模。

例 4.4　假设问题规模和处理器的数量固定，对于我们的机器节点中最低层次的高速缓存（即距离处理器最远、距离存储器最近的高速缓存），一个应用具有如图 4-4a 所示的扑空率和高速缓存尺寸对照的曲线。如果 C 是高速缓存的尺寸，这个曲线如何影响我们对用于评价机器的问题规模的选择？

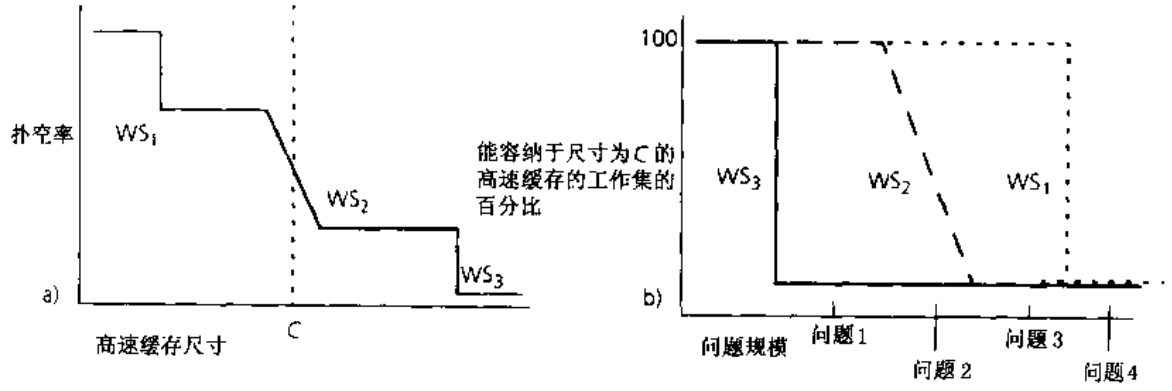


图 4-4　基于能容纳于高速缓存的工作集来选择问题的规模。在 a）中的图显示了对于固定的问题规模和所选定的处理器数量，扑空率和高速缓存尺寸对照的曲线。C 是所考虑的高速缓存或复制存储器的尺寸。这条曲线显示了三个拐点或者说工作集，两个定义得非常陡峭，另一个要缓和些。b）中的图表明，对于每一个工作集一条曲线描述了当问题规模提高时，它是否能容纳于尺寸为 C 的高速缓存。曲线的拐点表示了工作集合不再能被容纳的问题规模。可以看出规模为问题 1 的问题适合于 WS_1 和 WS_2 但是不适合于 WS_3。问题 2 适合于 WS_1，部分适合于 WS_2，但不适合于 WS_3。问题 3 只适合于 WS_1。问题 4 不适合于高速缓存中的任何工作集

解答：从图 4-4a 中我们可以看到对于所给出的问题规模（和处理器的数量），第一个工作集能容纳于尺寸为 C 的高速缓存，第二个工作集只能被部分容纳，第三个则不能被容纳。每一个工作集按照它们自己的方式随问题规模而扩放，这种扩放决定了在何种问题规模下，该工作集不再能够被尺寸为 C 的高速缓存所容纳，因此决定了应该选择什么样的问题规模才能覆盖各种有代表性的情况。实际上，如果曲线真的包含陡峭的拐点，那么可以画一种不同类型的曲线，此时每个重要工作集对应一条曲线。图 4-4b 中的曲线描述了当问题的规模变化时，工作集是否能被尺寸为 C 的高速缓存所容纳。如果出现在曲线拐点处的问题规模处于我们所决定的实际的问题规模范围之内，那么应该保证把这个拐点两侧的问题规模包括在我们的选择之内。不这么做的结果是，可能会失去与存储器或者通信体系结构相关的重要效应。在这个例子中，拐点两侧的曲线是平滑的。这一事实说明，如果对高速缓存我们关心的只是扑空率，那么为了这个目的，只需要在每个拐点的每一侧选择一种问题规模，剪裁掉

其他的部分[○]。■

例 4.5　对于方程求解器，工作集如何影响我们对问题规模的选择？

解答： 方程求解器最重要的工作集出现于当一个分区的两个子行能容纳于高速缓存以及当一个处理器的整个分区能放进高速缓存的时候。在这个简单内核中非常明确地定义了这两种情况。即使对于基于固有通信与计算的比率所选择的最大网格（2 K × 2 K），每个处理器的数据集尺寸仅仅是 0.5MB，所以这两个工作集合都能自如地容纳于高速缓存（如果使用了一个四维数组的表示，基本上就没有因冲突而产生的扑空）。因此，可能还需要选择更大一些的问题规模。要使两个子行构成的第一个工作集超出一个 1 MB 的高速缓存，这意味着一个子行要有 64 K 个点，所以对于 64 个处理器的机器，整个的网格是 64 K × $\sqrt{64}$或者说 512 K 行（或列）。在这种情况下，每个处理器的数据集是 32 GB，大得有点不符合实际。但是，使另外一个重要的工作集（即一个处理器的整个分区）不能被一个 1 MB 的高速缓存所容纳却是现实的。这就导致了或者有很多本地存储器流量，或者有很多人为通信（如果数据没有被合适地存放），我们将表示这种情况。可以选择一个问题规模，如每个处理器 512 × 512 个点（2 MB）或者说整体 4 K × 4 K 个点。这样做不会填满机器的存储器，所以可以再选择一种问题的规模，如整体 16 K × 16 K 个点或者说每个处理器 32 MB。现在有 5 种问题规模：256 × 256，1 K × 1 K，2 K × 2 K，4 K × 4 K 和 16 K × 16 K。■

第 4 步：使用空间局部性

假设以共享地址空间实现的方程求解器内核中用于表示网格的数据结构是一个二维数组。作为它的重要的工作集，一个处理器的分区在跨越不同网格扫描时，可能已经不在它的高速缓存中了。即使高速缓存的容量足够大，高速缓存冲突仍可能相当频繁，因为一个处理器分区的子行在地址空间中是不连续的。在任何一种情况下，如果工作集不能被高速缓存所容纳，那么将一个处理器分区分配到具有分布式存储器机器的本地存储器中是很重要的。主存分配的粒度是页，通常为 4 ~ 16 KB。如果一个子行的尺寸小于页的尺寸，合适的分配是非常困难的，而且可能会导致很多人为的通信。但是，如果一个子行的尺寸是页尺寸的几倍，分配则不成问题，几乎不会有什么人为的通信。这两种情况都可能是符合实际的，所以我们应该试着来表示这两种情况。如果页的尺寸是 4 KB，已经选择的前三个问题的规模具有小于 4 KB 的子行，因此它们不能被合适地分布；后两个问题有大于或者等于 4 KB 的子行，因此只要网格对齐页的边界，它们就能够被合适地分布。所以，无需为此而扩展问题规模的集合。对于以一个四维数组表示的网格，处理器的网格分区在地址空间中是连续的，所以当分区大到必须分配时合适的分配是容易的。

223 ~ 224

一个更加严格的空间局部性相互作用的例子出现在不同的程序和体系结构的相互作用中。一个被称为 Radix 的程序是一个将在本章后面介绍的排序程序，被称为伪共享的体系结构的相互作用已经在第 3 章定义了，在第 5 章将要作进一步讨论。但是，在这里先看看其结果，从而阐明我们在选择问题规模时考虑空间相互作用的重要性是有益的。图 4-5 表明了运行在高速缓存一致的共享地址空间机器上的程序在使用相同数量即 p 个处理器的情况下，对于两种不同的问题规模 n（对256 K个整数和1 M个整数进行排序），其扑空率是如何随

○ 如果我们也关心除了扑空率之外同样受到尺寸影响的高速缓存的其他行为方面的话，剪裁掉扑空率曲线的扁平区域则可能是不适当的。我们在第 5 章讨论高速缓存一致性协议的折中方案时将会看到一个这样的例子。

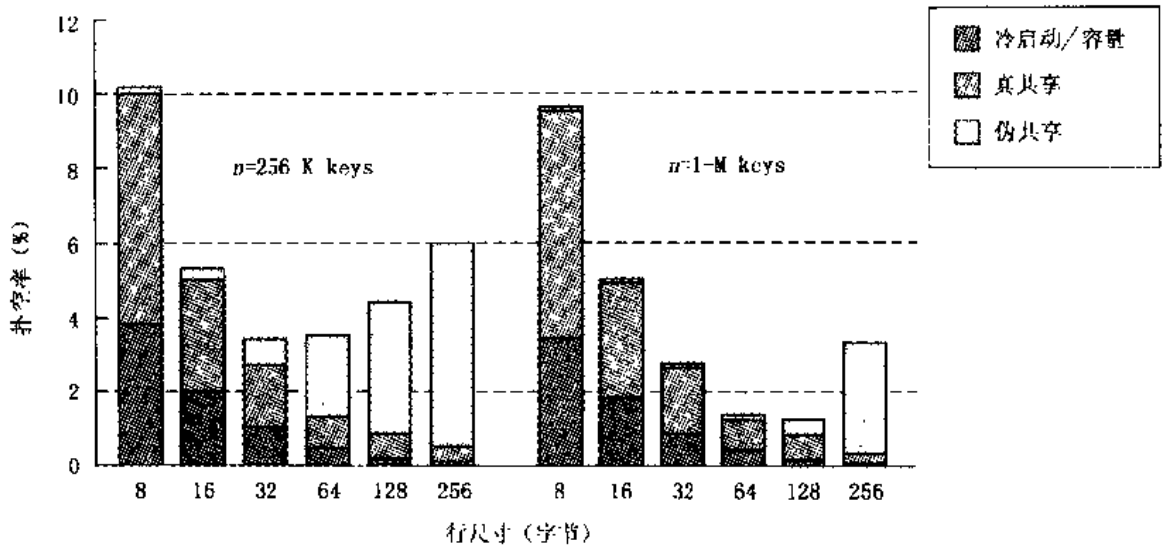


图 4-5　问题规模和处理器的数量对基数排序的空间局部性行为的影响。扑空率被分成冷启动/容量型扑空、真实共享（固有通信）扑空和由伪数据共享引起的扑空。对于给定的问题规模和处理器数量，当块的尺寸增加时，就会出现这么一个点，即文中讨论的临界率比块尺寸的多倍阈值要小，此时就会出现严重的伪共享。对于不同问题规模，这个阈值效应在不同块尺寸处发生。如果问题规模保持不变而处理器的数量变化时，也可以观察到类似的效应

225 高速缓存块的尺寸变化的。扑空率中的伪共享成分随着缓存块的尺寸而增加，当它变得显著时，会导致很多人为的通信破坏该应用的性能。如果给定机器的高速缓存块的尺寸，伪共享会不会破坏基数排序的性能，这取决于问题的规模（比较具有 64 个字节的块的直方条）。其结果是对于给定的高速缓存块尺寸，如果问题规模与处理器数量的比值小于某个阈值时，伪共享成分大；反之，当这个比值较大时，伪共享不显著。

很多应用在空间局部性与问题规模的相互作用中显示了这种阈值效应；在其他的应用程序中，特别是在很多非规则的应用程序如 Barnes-Hut 和 Raytrace 中，其数据结构和访问模式使得空间局部性不会随着问题规模的上升而增加很多。确认这种阈值的存在需要理解应用程序的局部性以及它和体系结构参数的相互作用，阐明评价中的一些微妙之处。

总结一下，简单的方程求解器说明了很多执行特征对问题规模的依赖，某些特征在和体系结构参数的相互作用中在阈值处呈现出拐点，而另一些则没有呈现。对于 $n \times n$ 的网格，p 个处理器，如果 n/p 的比率大，那么通信与计算比率就低，重要工作集不太可能被处理器的高速缓存所容纳，导致容量型扑空率高；但是即使是使用一个二维数组表示，空间局部性也很好。当 n/p 小时，情况则相反：会出现高的通信与计算的比率、差的空间局部性、伪共享（用二维数组表示），但几乎不存在本地容量型扑空。因此主要的性能瓶颈从第一种情况的本地访问转变为后者的通信。图 4-6 说明了对于 Ocean 应用程序的整体上的效应，Ocean 使用了类似于方程求解器的内核。

其他应用程序可能会呈现出对问题规模不同的特殊依赖性。尽管为评价机器而选择问题的规模并没有普遍适用的规则可循，方程求解器的内核只是一个很小的例子，但本章所给出的步骤提供了一个有用的方法，应该能保证从机器获得的结果并不是来自很容易从程序中去除掉的人为因素。如果我们要比较两台机器，选择能在两台机器上进行上述工作的问题规模是十分重要的。尽管要考虑的问题很多，实验表明，以一个应用程序评价规模固定的机器所

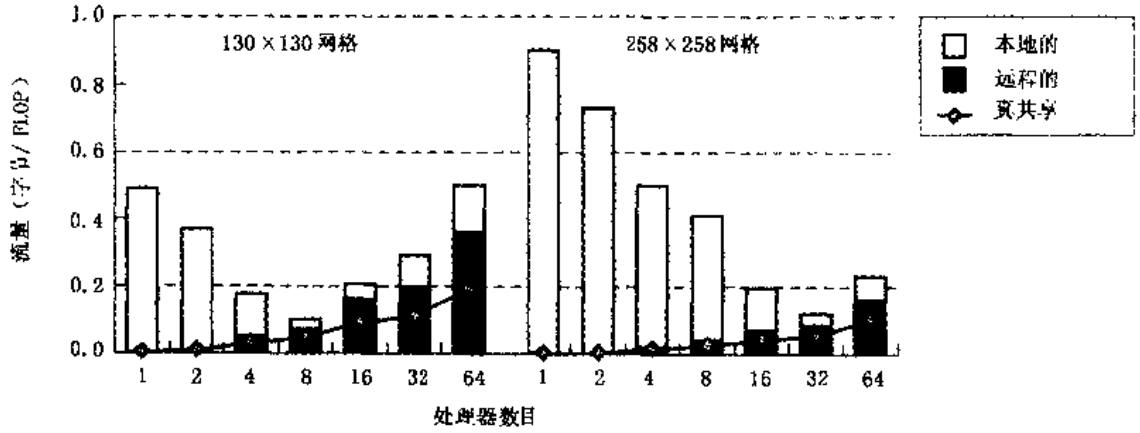


图 4-6　问题规模、处理器的数目、可被高速缓存容纳的工作集的效应。本图中显示了在共享地址空间中，Ocean 应用程序的存储器行为的效应。高速缓存扑空的流量（每次浮点运算或 FLOP 的字节数）被细分为本地的或者说节点内部的流量和远程的或者说经过网络的流量（即通信）。由真实数据共享（固有通信）所产生的流量也单独给出。远程流量随处理器的数目增加而上升，随着问题从小到大而减少。对于给定的问题规模，处理器数目增加时，工作集开始能被高速缓存所容纳，由本地扑空占主导地位转变为通信占主导地位。这个变化发生在较大问题规模和较大数量的处理器的情况下，因为工作集和 n^2/p 成比例。如果我们针对这两种问题规模，集中观察 8 个处理器的情况，我们可以看到对于小问题，流量主要是远程的（因为工作集能装入高速缓存），而对于较大的问题，流量主要是本地的

需要的问题规模的数目通常是相当小的，因为只有几个重要的阈值。

4.2.4　改变机器的规模

现在假设我们想要在处理器的数量变化时评价机器的性能。我们已经看到了在不同的扩放模型中如何扩放问题的规模，以及对由于并行性带来的性能改善应采用什么样的指标。剩下的问题是，在某种机器规模条件下，如何选择作为扩放开始点的问题规模。一个策略是从以前为固定数量的处理器所选择的问题规模开始，根据不同的扩放模型向上或向下扩放它们。我们可以将基本问题规模的范围缩小为三类：小、中、大，与三种扩放类型组合将会产生 9 组性能数据和加速比曲线。但是，可能需要注意当问题规模缩小时，它要保证能测试较小机器的能力。

另外一种策略是在一个单处理器上从一些精心选择的问题规模开始，然后在所有 3 个模型从下向上扩展它们，这里，选择 3 种单处理器问题的规模也是有道理的。小的问题应该是它的工作集能容纳于单处理器高速缓存的问题，这个问题在大机器上的问题约束（PC）扩放情况下不是很有用；但是在存储器约束（MC）扩放下或许甚至在时间约束（TC）扩放下它应该是没问题的。大问题应该是其重要工作集不能被单处理器的高速缓存所容纳的问题，如果这对于应用来说是切合实际的话。在 PC 扩放下，工作集合可能在某些点能够容纳于高速缓存（如果它随着处理器个数的增加而收缩）；而在 MC 扩放下，工作集不大可能被高速缓存所容纳，它可能不断地产生由容量型扑空造成的流量。大问题的一个合理选择是使它充满单个节点的几乎所有内存或者在节点上运行很长时间。因此，在 MC 扩放下，即使是在大系统中，它也会充满几乎整个内存。中等规模问题的选择是在大小两者之间做出一个明智的选择，如果可能它也要在单处理器上运行相当长的时间。一个突出的问题是，如何在不产生

超线性加速比问题的前提下，对于不能容纳于单个节点的内存的问题规模探索 PC 扩放。这里，一种解决方案是简单地选择这样一个问题规模，不是相对于单处理器，而是相对于多个处理器测量加速比；对于多个处理器而言，该问题确实能被存储器所容纳。

4.2.5 选择性能指标

在评价和比较机器时的一个重要的问题是应该使用的特定指标，正如没有适当地选择工作负载和参数很容易产生误导一样，没有用有意义的方法测量和表示结果也很容易得出错误的印象。总之，代价和性能都是比较机器和评价性能的重要指标。在评价机器随着资源（比如，处理器和存储器）的增加是否能很好地扩放时，我们关心的不仅仅是性能如何提高，而且也关心代价如何增加。即使加速比的提高比线性提高要小得多，如果运行程序所需要的资源的代价增加得没有加速比增加得那么快的话，那么使用较大的一些机器所花费的代价的确是值得的（Wood and Hill 1995）。总的来讲，一些对代价和性能的综合测量比简单的性能测量更合适。但是，我们可以分别测量代价和性能，代价在相当程度上依赖于市场，因此在这里主要关心的是性能测量的指标。

绝对性能和由于并行性产生的性能改善两者都是有用的指标，这里，我们考察一下使用这些指标来评价机器，特别是比较机器时的一些细致的问题并考虑其他一些基于处理速率（比如，每秒百万次浮点操作（megaflop））、资源使用、问题规模，而不是直接基于工作量和时间的指标的作用。某些指标显然是十分重要的，应该总是提供，而另外一些指标的使用却取决于我们以后要干什么以及我们所工作的环境。

1．绝对性能

对于系统的用户来说，绝对性能是他最关心的性能指标。假设执行时间是我们关于绝对性能的指标，可以用不同的方法测量时间。首先，对于一个工作负载，在用户时间和墙钟时间之间要做出选择。用户时间是机器花在执行工作负载上的时间，不包括系统行为和其他可能分时共享机器的程序。墙钟时间是指运行工作负载而流逝的全部时间，包括所有与此相关的行为。其次，存在的另一个问题是在程序的全部进程中使用平均执行时间还是最大执行时间。

因为用户最终关心的是墙钟时间，在比较系统的时候，我们必须测量和表示它。但是，如果其他用户程序（不只是操作系统）作为多道程序和该程序的执行相互打扰的话，那么墙钟时间并不能帮助我们理解性能瓶颈。注意，在这种情况下，那个程序的用户时间也不会很有用，因为与不相关进程的交错执行会破坏程序的存储器系统的相互作用以及它的同步和负载平衡行为。因此，不管我们是否为了增强理解而提供更详细的信息，我们总是应该提交墙钟时间，同时描述执行的环境（批处理程序或多道程序）。如果我们想要理解一个特殊应用的性能，我们应该在只有操作系统介入的情况下独立地运行它。

228

同样，因为直到最后一个进程结束时，并行程序才结束，正是到达这一点的时间很重要，而不是进程的平均时间。用平均时间的概念容易忽略不平衡。当然，如果我们真正想理解性能瓶颈，我们会愿意看一看所有进程的执行性态（或者至少一个采样）分解成不同的时间成分的情况（如图 3-12）。执行时间的组成成分说明了为什么一个系统比另外一个系统性能好，工作负载对于研究是否合适（例如，未受到负载失衡的限制）。

2. 性能的改进或者加速

对任何一个扩放模型测量加速比时的一个问题是加速比的分子，即单个处理器的性能，应该实际测量什么。我们有4种选择：

1）在并行机器的一个处理器上并行程序的性能。

2）在并行机器的一个处理器上同一算法的串行实现的性能。

3）在并行机器的一个处理器上对于同一问题最优的串行算法和程序的性能。

4）在公认的一台标准机器上最佳串行程序的性能。

1）和2）的区别在于，即使是在单处理器上运行并行程序也会产生额外的开销，因为它执行了同步操作、并行性管理指令或产生划分的代码或者甚至是为了省略这些操作所做的测试。这个额外开销有时候会相当显著。2）和3）的区别在于最佳串行算法的并行化可能无法或难以有效地实现，所以并行程序所使用的算法会有别于最佳的串行算法。

从用户的观点看，使用3）所定义的性能显然会导致比1）和2）更好和更加精确的加速比指标。但是，从体系结构设计者的观点看，在很多情况下使用定义2）是可行的。定义4）将机器的单处理器的性能融合进来，因此会产生和绝对性能相类似的比较指标。 229

3. 处理速率

一个经常引用的表现机器性能特征的指标是每单位时间内执行的计算机操作的数量(计算机操作与在应用层有意义的操作，比如事务处理或者化学结合不同)。经典的例子是用于浮点运算密集型程序的MFLOPS（每秒百万次浮点操作）和用于一般程序的MIPS（每秒百万条指令)。尽管它们在厂商的市场宣传材料中很是流行，但是已经有很多人写文章指出为什么它们不是好的通用性能指标，其基本的原因是，除非我们使用一个明确的、独立于机器的度量方法能够测量解决一个问题所需要的基本的FLOP或者指令的数目，而不是测量实际执行的FLOP或指令的数目，否则这些测量可以人为地膨胀：执行更多的FLOP，花费很长时间的低级蛮干型算法可能反而会产生更高的MFLOPS速率。实际上，我们甚至可以通过在代码中插入无用但是廉价的操作来提高这样的指标。如果需要的操作数目是明确已知的，那么使用这些基于速率的指标和使用执行时间没有什么不同。MFLOPS的其他一些问题包括：不同的浮点操作有不同的代价；即使在浮点操作密集型的应用中，现代的算法也要使用有很多整数操作的精巧的数据结构；这些指标还受到遗留的错误使用方式（比如，公布硬件的峰值速度而不是实际的应用达到的速度）的拖累。如果使用得合适，像MFLOPS和MIPS这样基于速率的指标对于理解基本硬件的能力还是有用的，但是若把它们作为机器性能的主要标志使用时，我们应该非常谨慎。

4. 利用率

体系结构设计者有时会以能否让处理引擎保持忙碌，不断地执行指令而不因为各种额外开销而停顿下来，作为衡量他的设计是否成功的标准。但是，现在有一点应该很清楚，处理器的利用率不是用户感兴趣的指标，也不是一个良好的惟一性能指标。它也可以被人为地扩大，有利于较慢的处理器，对于最终性能或者性能瓶颈也不能提供多少有用的信息。但是，作为决定是否要开始深入地探究程序或机器中的性能问题的出发点，它可能还是有用处的。同样的论点也适用于其他资源的利用率。利用率对于决定一台机器的设计是否在各种资源间平衡以及决定瓶颈所在是有用的，但是对于性能的测量和比较没有多少用处。

5. 问题规模

另外一个有趣的指标是能够获得特定的并行效率的给定应用的最小问题规模，它被定义
230 为加速比除以处理器的数量（在给定的扩放模型下）。由于并行性产生的额外开销通常随着问题规模而相对减小，改进的通信体系结构的一个好处是运行较小问题的能力提高。在某种意义上，当处理器数量增加时保持并行效率不变产生了一种新的扩放模型，我们称之为效率约束扩放。当然，这个指标必须要小心使用，因为容量效应可能主导了通信的差别，而小问题可能无法突出系统的重要方面。并行效率是有用的，但不是一个通用的性能指标。

6. 性能改善的百分比

人们有时使用的一个用于评价因某个体系结构特性带来的性能改善的指标是执行时间或该特性所提供的加速比被改善的百分比。如果没有提到原来的并行性能（如原来的加速比），这种指标会在并行系统中起误导作用。比如，在一个 1 024 个处理器的系统中把加速比从 400 提高到 800 和把加速比从 1.1 提高到 2.2 两者的改进百分比相同，但是后者对于一个有 1 024个处理器的系统来讲，是没有什么意义的。如果问题的规模是导致加速比差的原因，那么用提高问题规模来产生很好的加速比却经常会严重地降低由特性所获得的改进。同样，这个指标是有价值的，但是必须由其他指标予以补充以避免误导。

总之，代价和性能都是需要考虑的重要因素。从用户的观点来比较机器，最感兴趣的性能指标是墙钟执行时间。但是，从系统设计师和力图理解程序的性能的编程人员，或者甚至从那些对机器性能有更为普遍性兴趣的用户的观点看，最好是看一下执行时间和加速比。这两种指标都应该在任何研究的结果中提交。理想的做法是，应该像在 3.4 节中讨论的那样，把执行时间分解成主要的组成成分。为了理解性能瓶颈，以每个进程为基础观察这种执行时间的成分分解，或将其看作平均及进程间分散程度的某种度量（简单的平均是不够的）是非常有用的。在评价变化对于通信体系结构的影响，或者比较基于相同底层节点的并行机器时，诸如实现一定目标所需的最小问题规模，这样的基于规模或者基于配置的指标是有用处的。像 MFLOPS、MIPS 和处理器利用率这样的指标可以用于特定的目的，但是仅用它们来表示性能则需要关于表示者的知识和完整性的很多假设，它们也受到遗留的不当使用方式的拖累。

4.3 对一个体系结构概念或设计权衡的评估

假设你是一个计算机公司的系统设计师，准备设计下一代的多处理器系统。你萌发了关
231 于体系结构的一个新的概念，要决定是否把它包含在将要设计的机器中。对于上一代机器的性能和瓶颈，你可能有丰富的信息，这实际上可能是促使你进一步研究这个概念的最初动因。但是，你的概念和所拥有的数据并不全是新的。从上一代机器问世以来，从集成的层次到多处理器使用的高速缓存的规模以及组成结构，技术已经发生了变化。你将要使用的处理器可能不仅仅是快得多，而且也要复杂得多（比如，动态调度的四路指令发射和静态调度的单指令发射之比较），它的新的能力可能会影响你现在的新概念。操作系统也可能发生了变化，同样编译器、甚至是有意义的工作负载也都会发生变化，而且这些软件成分可能在机器实际建造和售出之前还会进一步变化。你所感兴趣的特性要求的硬件，特别是其设计时间的代价相当昂贵，而且你还要赶在截止期之前完成它。

在如此众多的巨大变化下，你所拥有的用于做出有关性能和代价决定的数据的有效性是令人怀疑的。你最多可以把它们和你的直觉揉合在一起使用，做出有见解、有素养的猜测。

但是如果特性的代价高，你可能希望再多做一些事情。你所能做的是构造一个能模拟你的系统的模拟器。你还需要将其他所有的东西——编译器、操作系统、处理器、技术和体系结构参数——固定在它们希望的配置上，只用你所感兴趣但尚不具备的特性来模拟系统，然后提交结果，判断它对性能的影响。然后，你或许应该检查一下你已经固定了的某些方面的敏感性，但是这些可能不容易预测。

为并行系统构造精确的模拟器是困难的。很多在扩展的存储器层次结构上的复杂交互很难被正确地模拟，特别是那些与资源占用和竞争有关的交互更是如此。处理器本身变得更加复杂，精确的模拟要求它们也必须在细节上被模拟。但是，即使你能够设计一个非常精确的模拟器来模拟你的设计，你仍然有一个大问题，即模拟是昂贵的，它要占用大量的存储器和时间，特别是当模拟大的问题和机器时。这意味着你不能模拟真实大小的问题和机器规模，你必须设法适度地缩小模拟的规模。

甚至连你的技术参数也可能是不固定的，你从一张白纸开始，想知道在不同的技术假设下你的概念工作得如何。现在，除了前面的工作负载、问题规模、机器规模几个坐标轴外，机器参数也可以变化。这些参数包括本地存储器层次结构中各层的规模和组成结构；分配、通信和一致性的粒度；时延、占用率和带宽等通信体系结构的性能特征。这些参数和那些工作负荷的参数一起产生了你必须遍历的巨大的参数空间。模拟的高代价和种种限制使得剪裁这个设计空间而又不失去太多的覆盖更加的重要。本节将讨论在模拟研究中，选择参数和剪裁设计空间的方法上的考虑，使用一个特别的评价作为例子。首先，让我们快速地看一下多处理器的模拟。 232

4.3.1　多处理器的模拟

尽管是要模拟多个进程和处理节点，模拟本身却可以在一个单处理器上运行。一个叫做访问生成器的部件扮演了并行机器中处理器的角色。它模拟了处理器的行为，产生对存储器的访问（同时带有表明访问是来自哪个处理器的进程标识）或者对模拟存储器系统和互连网络的模拟器的命令（比如发送或接收，见图 4-7）。如果是在单处理器上运行模拟，被模拟的不同进程分时共享该单处理器，由访问生成器进行调度。一个调度的例子是每当进程发出对

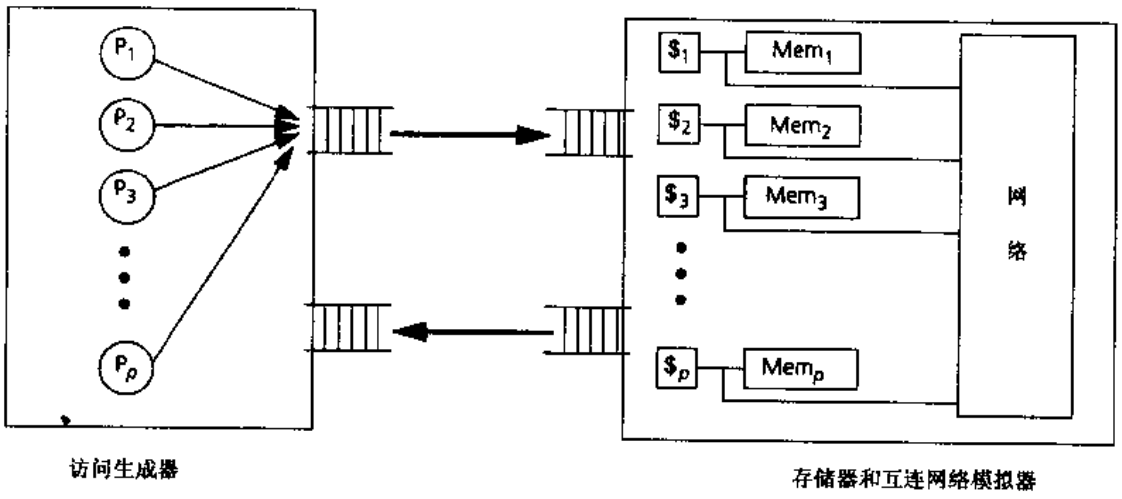


图 4-7　执行驱动的多处理器模拟。被模拟的处理器发出对存储器系统模拟器的访问，存储器系统模拟器模拟扩展的存储器层次结构并且向被模拟的处理器（访问生成器）反馈定时信息。$1，$2 等表示高速缓存

存储器系统的访问时就让它退出运行，允许另一个进程开始运行，直到那个进程产生了它的下一次访问。另外一个调度的例子是在每个模拟的时钟周期中重新调度一次进程。存储器系统模拟器模拟不同处理节点上所有的高速缓存、主存以及互连网络本身，它对数据通路、时延和竞争的模拟可以是任意复杂的。

访问生成器（处理器的模拟器）和存储器系统模拟器之间的耦合可以按照不同的方式组织，这取决于模拟所需要的精度和处理器模型的复杂程度。一个选择是轨迹驱动的模拟。在这种情况下，首先通过在一个系统上运行并行程序获得每个进程所执行的指令的轨迹，运行并行程序的这个系统可能与被评价的系统不同。这个轨迹替代了访问生成器：来自轨迹的指令被送入模拟目标多处理器扩展的存储器层次结构的模拟器。这里，耦合或者信息流只是单方向的：从访问生成器（这里只是一个轨迹）到存储器系统模拟器。

模拟的另一个更加流行的形式是执行驱动的模拟，它提供双向的耦合。在执行驱动的模拟中，当存储器系统模拟器接收到来自于访问生成器（现在是一个程序，而不是一个预先决定的轨迹）的访问或者命令时，它模拟在扩展存储器层次结构中所经过的访问路径，包括和其他访问的竞争，并向访问生成器返回满足该访问所花费的时间。和关于公平性及保持同步事件的语义的考虑一起，访问生成器程序，使用这个信息来决定下一次调度哪一个被模拟的进程以及何时从那个进程发出下一条指令。因此，从存储器系统模拟器到访问生成器发生了反馈，像在一台真实的机器中一样；这种反馈影响着后者的行为提供比轨迹驱动的模拟更高的精度。为了允许在事件和访问的模拟中最大限度的并行性，存储器系统和网络的大部分组成部件也被模拟成由模拟器调度的独立的相互通信的线程。模拟器维护一个全局的模拟时间，即被模拟的机器已经看到的虚拟时间，而不是模拟器本身已经运行的真实时间。这正是我们在决定被模拟的体系结构中工作负载的性能时要查看的时间和赖以做出调度决策的时间。除了时间以外，模拟器通常保存大量的关于各种有意义的事件的统计数据，这就提供了丰富的详尽的性能信息，这些信息在真实的系统中虽不是不可能获得，但至少也是难以得到的。但是，模拟的结果可能会因为缺少可信性而遭到玷污，因为它毕竟只是一次模拟。当必须要模拟复杂的、动态调度的、多指令发射的处理器时，精确的执行驱动的模拟也是要困难得多。习题 4.9 讨论了模拟技术中的一些折中。

4.3.2　缩小模拟的问题和机器参数的规模

如果知道了模拟是用软件实现的，涉及很多被非常频繁地重新调度的进程或者线程（精度要求越高，重新调度越频繁），模拟非常昂贵这一事实就并不令人感到惊奇了。对模拟本身的研究正在进行，使其加速，使用硬件仿真而不是软件模拟（Reinhardt et al. 1993; Goldschmidt 1993; Barroso et al. 1995），但是所取得的进展并不是那么显著，不能改变必须大大缩小参数的事实。

关于缩小问题和机器参数的一个棘手的问题是，我们希望运行较小问题的缩小规模的机器能代表运行较大问题的全规模的机器。不幸的是，没有保证实现这一点的规则可循。不管怎样它是一个重要的问题，因为这是大多数对体系结构折中的评价中存在的现实。我们至少应该理解这种扩放的局限性，认识哪些参数能被可信地缩小，而哪些不能，发展出一些能帮助我们避免主要误区的指导方针。让我们先来考察一下缩减问题规模和处理器的数量，然后解释一些和低层次的机器参数相关的难点。再说一遍，为具体起见，我们仍然将注意力集中

在高速缓存一致的共享地址空间的通信抽象上。

1．问题参数和处理器数量

先考虑问题参数。我们应该先找到那些如果存在就会严重影响模拟时间，但是却很少影响和并行性能相关的执行特征的问题参数。一个例子是很多科学计算，比如 Ocean 或者甚至 Barnes-Hut 所执行的时间步的数量，或者是在简单的方程求解器中的迭代次数。在时间步之间，操作的数据值会起很大的变化，但是行为特征不会变化太大。在这种情况下，我们的模拟可以只运行少数几个时间步[⊖]。

不幸的是，很多应用程序的参数要影响和并行性相关的执行特征。当缩小这些参数时，我们也必须减少处理器的数量，因为不这么做的话，我们可能会得到完全没有代表性的行为特征。但是，用具有代表性的方法做到这一点是很困难的，因为我们面临着很多约束，这些单个的约束都难以满足，要让它们彼此协调或许是根本不可能的。这些约束包括：

- 保持程序各阶段中花费的时间分布。用于执行不同类型的计算，比如 Barnes-Hut 中树的构造和力计算阶段，所花费的相对时间量最可能随着问题和机器的规模变化。
- 保持关键的行为特征。这包括通信与计算的比率、负载平衡、时间和空间局部性，它们全都会以不同的方式扩放！
- 保持应用程序参数之间的规模扩放关系。
- 保持竞争和通信的模式，这特别难，因为突发性是很难预见和控制的。

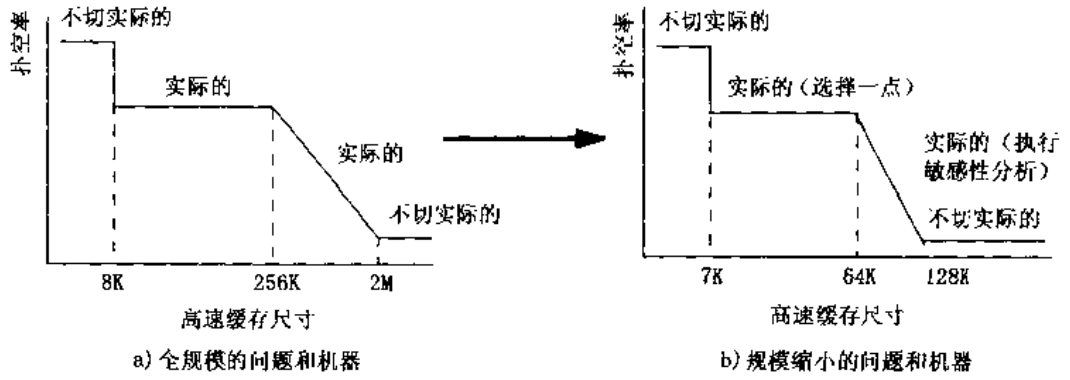
一个更加现实的目标是，当缩小规模的时候，不是保持真正的代表性，而是至少覆盖与研究最关心的行为特征相关的实际运行点的一个范围，避免不切实际的场景。因此，不能说缩小规模的模拟是有定量代表性的，但是可以用它们来获得见识和粗略的估计。有了这个更 235
加适度的目标，让我们假设我们已经以某种合理的方法缩小了应用程序的参数和处理器的数量，现在来看看如何扩放机器的其他参数。

2．其他的机器参数

当问题与机器的规模缩小时，它们与低层次的机器参数的相互关系与全规模问题时不同。所以必须小心地调整这些参数。

考虑高速缓存或者复制存储器的尺寸。假定我们为方程求解器内核所能模拟的最大的问题和机器的配置是 512 × 512 的网格，在 16 个处理器（即每个处理器 128 KB）上运行。如果不缩小每个处理器的 1 MB 高速缓存的话，将无法表示重要工作集无法容纳于高速缓存的情况。关于扩放高速缓存的关键点在于，必须理解在所讨论的各个实际或不实际的工作点上，相关的工作集如何扩放（如图 4-4 所示），在此基础上对高速缓存进行扩放。一般来说，完全不调整高速缓存的大小或简单地根据数据集的尺寸或问题的尺寸按比例缩小高速缓存是不合适的，因为高速缓存的大小是与工作集的尺寸而不是与数据集或问题的尺寸最密切相关。例 4.6 和图 4-8 说明了如何根据给定问题的规模和机器的尺寸选择高速缓存的大小。我们还应该保证我们所模拟的高速缓存不会变得非常小，因为这样做会受到非代表性的映射和人为

⊖ 当然，我们现在应该从测量中忽略初始化和冷启动的阶段，因为在减少了时间步之后，它们在运行中的影响就要比在实际情况下大得多。如果我们希望在一段长的时间内行为会发生显著的变化，正像在 Barnes-Hut 或者其特征变化剧烈的应用中那样，那么，我们就可以从一台具有我们正在模拟的问题配置的真实机器的执行中周期性地卸出程序的状态，并且以这些卸出的状态作为输入数据集，启动几个样本模拟（在每个样本模拟中，不测量冷启动）。也可以用其他的采样技术来降低模拟的代价。



a) 全规模的问题和机器　　b) 规模缩小的问题和机器

图 4-8　为缩小的问题和机器选择高速缓存的尺寸。a）基于我们对工作集的尺寸及扩放的理解，我们首先决定工作集曲线的哪些部分对于在具有全规模高速缓存的机器上运行全规模问题是实际的。b）然后我们计划或测量在当前模拟的较小规模的问题和机器上工作集曲线看起来是怎样的，剪掉相应的不实际部分，对实际的部分选取有代表性的工作点（高速缓存的尺寸），方法与例 4.4 所讨论的类似。对于不能被剪除的部分，我们可以进行必要的敏感性分析

碎片的影响。除处理器高速缓存之外，类似的论点也适用于复制存储器，包括那些仅保存通信数据的存储器。

例 4.6　在 Barnes-Hut 应用中，假定运行 $n = 1$ M 粒子的全规模问题时最重要的工作集的尺寸是 150 KB，又假定目标机的每个处理器有 1 MB 高速缓存，而你只能模拟具有 $n = 16$ K个粒子的执行情况。将高速缓存的尺寸按数据集的大小成比例地缩小是否合适？你如何选择高速缓存的尺寸？

解答：回忆在第 3 章中 Barnes-Hut 问题的最重要的工作集按 $\log n$ 扩放，这里 n 是粒子的数量并与数据集的尺寸成比例。150 KB 的工作集可以很容易地放入目标机的 1 MB 高速缓存。由于其增长速率缓慢，在真实问题中工作集总是可以存放在高速缓存中。如果在模拟中按数据集的大小成比例地缩小高速缓存的尺寸，我们得到的高速缓存的尺寸将是 1 MB × 16 K/1 M，或者说 16 KB。而被缩小了的问题的工作集的尺寸是

$$150\ \text{KB} \times \frac{\log 16\text{K}}{\log 1\text{M}}$$

或者说 70 KB，这显然不能放入缩小后的 16 KB 的高速缓存。所以，这种形式的高速缓存的扩放产生了不能代表真实情况的工作点。因为我们希望在实际中工作集能放入高速缓存，我们应该选择足够大的高速缓存尺寸，从而总是能容纳工作集。■

当我们涉及到存储器层次结构的更低层次的参数时，对它们进行有代表性的扩放变得愈发困难。例如，扩放和高速缓存的相联度之间的关系非常难以预测，通常我们所能做的是不去改变其相联度。这样做的主要的危险在于当高速缓存的尺寸缩到非常小时，仍保持一个直接映射的相联度非常容易受到映射冲突的影响，这在全规模的高速缓存不会发生。除非存在着近乎完美的空间局部性，否则与其他一些存储器和通信体系结构的组织结构参数，如数据分配的粒度、传输和一致性的粒度等的相互关系也是复杂而且不可预料的。但是保持这些参数不变在很多情况下会导致严重的、非代表性的人为效应。在本章的习题中我们将看到一些例子。最后，当通信的频度和模式改变时，在保持代表性的前提下对时延、占用率和带宽等性能参数的适当缩减也非常困难。

总之，模拟的最好途径是尽可能运行真实规模（如果规模不太大的话）的问题。当需要缩小规模时，为了保证能覆盖重要类型的工作点，我们应该注意那些已经讨论过的指导方针和可能存在的误区，同时在推广结论时要小心。降低规模的可信度依赖于我们对应用的理解。一般来说，对仅仅要理解某些体系结构特性是否有益而言，使用缩小规模的方案还是可以的，但是如果要力图用它们来得出关于全规模情况的精确的定量结论是危险的。 236 ~ 237

4.3.3 处理参数空间：评价举例

现在考虑在通用的场景下试图评价某个概念时所产生的巨大参数空间的问题。为了使讨论具体化，考察一个在模拟时我们可能进行的实际的评价。再次假定一台具有高速缓存一致特性的共享地址空间的机器，其存储器在物理上是分布的。其缺省的通信机制是通过取和存实现的以高速缓存的块为单位的隐式的通信，但是我们希望探索是否能以尺寸较大的消息作为通信单位，降低端点的通信开销的影响以及通信的延迟。所以我们希望了解对这样的体系结构增加显式地发送较大的消息的能力的效果，这种能力叫做*块传输*，程序除了使用以高速缓存块为单位的标准传输机制之外还可以使用块传输（从而合并了共享地址空间和消息传递这两个编程模型）。例如，在方程求解器中，进程可以用单个块传输向其相邻的进程发送它的分区的完整的边界子行或子列。

在为这样的评价选择工作负载时，至少应该选择一些其通信可以被组织成较大的消息的负载，例如方程求解器。更为困难的问题是如何覆盖参数空间。我们的目标是三重的：

1）*避免不切实际的执行特征*。应该避免那些能导致不真实行为的特征，即在机器的实际使用中不会碰到的行为的参数组合（或运行点）。

2）*获得对真实执行特征的良好覆盖*。应该尽量保证那些在实际应用中会出现的重要特征能被表示出来。

3）*剪裁参数空间*。即使是在参数值的实际子空间内，为了在不损失多少覆盖度的前提下节约时间和资源，同时也为了决定何时需要做直接的敏感性分析，也应该基于对应用的理解，在允许的情况下尽量剪去工作点。

我们能根据研究的目标、技术（或特定硬件构造模块的使用）对参数的限制和对参数相互作用的理解来剪裁参数空间。

让我们用方程求解器为例来体验一次选择参数的过程。虽然应该对参数逐个考察，但在较晚的阶段出现的问题可能迫使我们重新考虑较早时所做出的决定。首先选定问题的规模和处理器的数量，因为这些是最受模拟资源的限制的。

1．问题规模和处理器数量

在选择问题的规模和处理器数量时应该考虑程序的固有的特征，这些特征在关于实际的机器的评价和缩减模拟的规模的讨论中已经涉及。例如，如果问题足够大从而使通信与计算之比非常小的话，那么块传输无助于改善整体的性能；如果问题足够小，从而使负载失衡成为主要的瓶颈时，采用块传输也无济于事。 238

现在把方程求解器的问题规模固定为 514×514 的网格，处理器的数量为 16，然后考察如何选择其他参数。

2．高速缓存/复制的尺寸

按照惯例，我们根据对工作集曲线的了解来选择高速缓存的尺寸。如果给定了工作集曲

线以及工作集如何扩放的知识，则为给定的问题选择高速缓存的过程与例 4.7 所讨论的根据给定的高速缓存尺寸选择问题规模的过程相类似。

例 4.7　图 4-9 给出了方程求解器的良好定义的工作集。在这种情况下你如何选择高速缓存的尺寸？

解答：虽然重要工作集的尺寸通常取决于应用的参数和处理器的数量，但它们的性质和工作集曲线的形状并不随这些参数而变化。因为我们知道在方程求解器中各个重要工作集的尺寸以及它们如何随参数而变化，如果我们也知道对于目标机而言真实的高速缓存尺寸的范围，那么我们就能够了解 1）期望工作集在实际情况下能容纳于高速缓存是否是不切实际的；2）期望工作集不能容纳于高速缓存是否是不切实际的；3）期望工作集在参数值的某些实际组合下能容纳于高速缓存，而在另一些组合下不能被容纳[○]，这种期望是否是切合实际的。所以我们能分辨曲线的拐点之间哪些区域代表了实际的情况，哪些不能代表。对于给定的问题的规模和处理器的数量，我们可以使用（固定的）工作集曲线来选择能避免非代表性区域的高速缓存的尺寸，覆盖代表性的区域并通过从中选择单一高速缓存尺寸来剪裁扁平的区域（如果我们关心的只是高速缓存的扑空率）。■

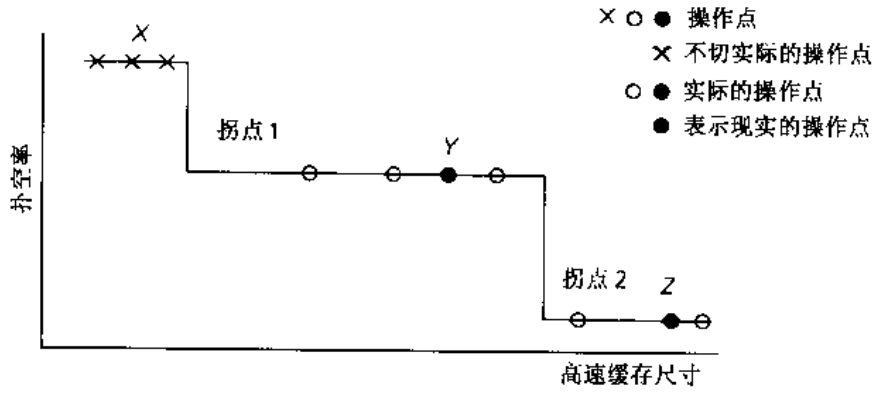


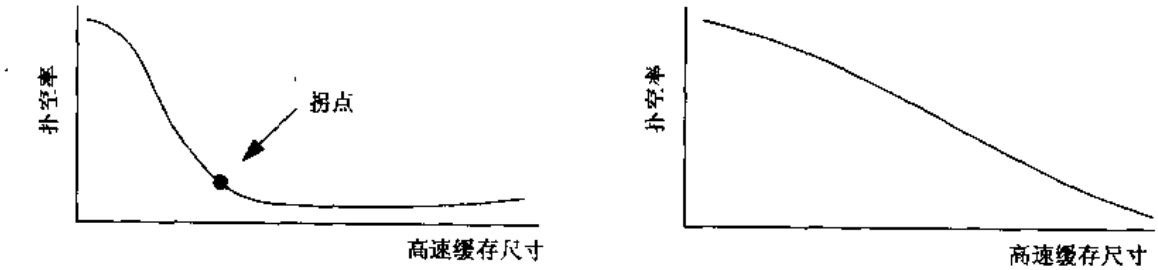
图 4-9　为使用方程求解器内核的评价来选择高速缓存的尺寸。拐点 1 大致对应一对 B 或 $n/\sqrt{p}$ 个元素的子行，取决于格的遍历是否被阻塞（对于 $B \times B$ 的块尺寸）。拐点 2 对应于一个处理器上的矩阵的分区（即数据集 n^2 除以 p）。后者的工作集根据 n 和 p 能否被高速缓存所容纳，所以 Y 和 Z 两者都是真实的工作点，必须被表示。对于第一种情况的工作集，如果遍历未被阻塞，它就不能被高速缓存容纳，但是在实际中我们知道有大的二级高速缓存，这种情况不大可能发生。如果遍历被阻塞，选择块尺寸 B 从而使第一种情况的工作集总是能被高速缓存所容纳。所以工作点 X 代表了不切实际的区域，因此被忽略。分块矩阵计算的情况在这方面与此类似

一个重要工作集是否能被容纳于高速缓存以一种有意思的方式严重影响块传输所带来的好处，其效应取决于工作集是否包含本地或非本地分配的数据。如果它主要由本地数据组成，正如数据适当分布的方程求解器的情况，但是它又不能被高速缓存所容纳，处理器将由于在本地存储器系统上受阻而花费更多的时间。其结果是通信时间变得相对地不重要，块传输没有多大帮助（块传输的数据对节点的本地流量的打扰更多，产生竞争）。然而，如果工作集主要包含非本地数据，我们将获得正面的效应：如果它们不能被高速缓存所容纳，就会有更多的通信，因此块传输就有更大的机会帮助性能的改善。

239

○ 给定尺寸的工作集能否被高速缓存所容纳除了取决于高速缓存的尺寸外，还取决于高速缓存的相联度和块的尺寸，但是如果我们假设至少 2 路的相联度的话，它在实际中通常不是主要的问题（我们在较晚些时候将看到这一点），所以我们现在忽略这些效应。

当然，工作集曲线并不总是包含由陡峭定义的拐点分隔的相对扁平区域。假如区域中有拐点，但是被拐点分隔的区域不是扁平的（见图 4-10a），若知道哪些区域是不切实际的，就能像以前那样剪切整个的区域。但是，如果区域是切合实际的但不是扁平的或在整个数据集能被高速缓存容纳之前不存在任何拐点（如图 4-10b），那么必须求助于敏感性分析，挑拣出靠近极端的点以及其间的某些点。再强调一下，适当的分析要求我们很好地理解应用的关键特征。



a) 拐点左边部分不是扁平的　　b) 对实际的高速缓存不存在扁平区域

图 4-10　不含被扁平区域所分隔的陡峭拐点的扑空率和高速缓存尺寸对照的曲线

剩下的问题是如何决定拐点的尺寸和拐点之间工作集曲线的形状。在简单的情况下，我们可以解析地完成这一任务。但是，算法复杂，常数因子难以预测，高速缓存的尺寸和相联度的效应难以分析。在这样的情况下，可以通过在给定问题规模和处理器数量的条件下测量（模拟）不同的高速缓存尺寸来获得曲线。所需要的模拟相对来说代价不大，因为工作集的尺寸并不依赖于详细的与定时相关的因素，例如时延、带宽、占用率和竞争，所以这些因素并不需要仔细地加以模拟（或根本无需模拟）。工作集如何随问题的规模和处理器的数量而变化也能够被适当地加以分析或测量。幸运的是，对增长速率的分析通常比预见常数因子要容易。而且，如果高速缓存足够大且结构合理的话（不是直接映射的高速缓存），像块尺寸和相联度这样低层次的问题通常也不会改变工作集（Woo et al. 1995）。 240

3．高速缓存块的尺寸和相联度

除了问题规模、处理器数量和高速缓存的尺寸之外，高速缓存的块尺寸是决定块传输收益的另一个重要参数，但是，问题要稍微复杂一些。对于具有良好空间局部性的程序而言，大的高速缓存块本身的作用就类似于小的块传输，使得显式的块传输相对不那么有效。另一方面，如果空间局部性不好，那么当使用读和写通信时，由大高速缓存块引起的额外流量（由于碎片或伪共享）会消耗掉比必须消耗的多得多的带宽。对于块传输来说，差的空间局部性是否也会浪费带宽则取决于块传输的实现是将整个高速缓存块还是只将所需要的字经由网络流水传送。注意，块传输本身增加了带宽的需求，因为它要在较短的时间内完成相同量的通信（如果能做到的话）。所以，如果块传输是以高速缓存块流水传输实现的，而且空间局部性不好的话，当可用的带宽有限时，使用块传输可能非但无益反而有害，因为它可能增加对可用带宽的竞争。

幸运的是，由于当前技术的约束或可用于构造系统的模块的限制，通常能够限制高速缓存块尺寸增加的范围。比如，几乎所有的现代微处理器都支持 32～128 字节的高速缓存块，而我们可能已经选择了具有 64 字节高速缓存块的微处理器。当问题规模和高速缓存块尺寸的相互作用产生阈值时（比如，前面提到的基数排序的例子），我们应该保证覆盖阈值的

两侧。

高速缓存相联度的影响的程度却是很难预见的，实际的高速缓存的相联度一般较小（通常最多4路），所以能考虑的选择的数量也少。如果必须选择单一的相联度的话，最好避免使用直接映射高速缓存（至少在远离处理器的高速缓存层次结构的最底层是这样），除非知
241 道所涉及的机器确实具有这样的高速缓存。

4．通信体系结构的性能参数

在讨论了整个存储器层次结构的组成结构参数之后，让我们考虑一下通信体系结构的关键性能参数，即额外开销、网络延迟或通过时间以及它们如何影响块传输的收益。我们应该基于我们对所涉及的真实系统的期望来选择这些参数的基础值，而这种理解有助于我们决定哪些参数应该变化及如何变化。

一次高速缓存块的交换（扑空时）的额外开销成分越高，以较大的高速缓存块传输来构造通信从而降低额外开销就越重要。只要开始一次高速缓存块传输的额外开销不会高得抵消了收益，这样做总是对的，因为显式地启动一次块传输的额外开销可能比隐式地启动高速缓存块传输的额外开销要大。

出于同样的考虑，节点间的网络通信时间越长，通过大块传输获得的好处越大（对这一点有限制，我们在第11章详细考察块传输时会讨论它）。改变时延通常并不能产生出现拐点或阈值点的效果，所以为了考察时延的可能的范围，我们必须通过在范围内选择几个点进行敏感性分析。在实践中，我们通常根据所涉及的机器的目标时延来选择时延；比如，紧耦合的多处理器的时延一般比局域网上的工作站的时延要小得多。

可用带宽也是我们的块传输研究的一个重要问题。带宽也呈现很强的拐点效应，它实际上是饱和效应：或者有足够的可用带宽满足应用的需要，或者不能满足。如果能够满足，那么可用带宽是4倍还是10倍于需要都没什么关系。所以我们能够挑选一种小于需求的带宽和一种远高于需求的带宽。因为块传输研究对带宽特别敏感，我们也可以选择靠近边界线的带宽。在选择带宽的值时，我们应该仔细地考虑应用对带宽需求的突发性，虽然应用整体上的平均带宽需求可能不大，但是在它的突发通信期间仍然可能使较高的带宽饱和，导致竞争。

5．重新修改选择

最后，我们可能经常会根据较晚的时候考虑的参数的相互作用修改我们早期做出的参数值的选择。比如，如果由于缺少模拟时间或资源，不得不使用较小的问题规模，可能试探使用非常小的高速缓存尺寸来表示重要工作集无法被高速缓存所容纳的真实场景。但是，选择非常小的高速缓存可能导致严重的人为效应，特别是如果我们使用直接映射高速缓存或大的
242 高速缓存块尺寸的话（因为这将导致高速缓存中块的数量很少，从而产生很多碎片和映射冲突）。所以我们应该重新考虑我们为了表示这种情况所做出的问题规模和处理器数量的选择。

4.3.4　小结

前面的讨论说明，如果我们没有充分覆盖参数选择空间的话，评价研究的结果可能误导：我们可以很容易地选择参数和工作负载的一种组合（例如，相对较小的问题规模、大的高速缓存和小的高速缓存块尺寸），它能证明块传输这样的特性可以带来性能上的好处；我们也可以同样容易地选择另外的组合证明块传输没有优点。所以，在体系结构研究中综合可

靠的方法性的指导方针并且理解硬件和软件的有关相互作用是非常重要的。

尽管存在许许多多的相互作用，幸运的是我们能找到一些参数和性质，它们处于足够高的层次，能够推理它们不依赖于机器较低层次的定时细节，而且应用的关键行为特征又依赖于这些参数和性质。我们应该保证覆盖关于这些参数和性质的真实工作区域，即应用参数、处理器的数量、工作集和高速缓存/复制尺寸间的关系（也就是说，重要工作集是否能容纳于高速缓存）。基准测试程序集应该对它们的应用提供基本的特征，比如并发性、通信计算比、数据局部性以及它们对这些参数的依赖性，这样体系结构设计师无需再生成它们（Woo et al. 1995）。

在应用参数和体系结构参数的相互作用中寻找拐点和扁平区域也是重要的，因为这对覆盖和剪裁特别有用。最后，研究的高层次的目标和约束也能帮助我们剪裁参数空间。

关于工作负载驱动的评价的方法问题的讨论到此结束。在本章的剩余部分，将介绍本书中最常使用的其余的并行负载。它也说明了所有工作负载与方法有关的基本特征。

4.4 说明工作负载的特征

本书大量使用工作负载来定量地说明所讨论的体系结构上的折中，并评价用于开展案例分析的机器。主要为支持一致性共享地址空间的通信抽象而设计的系统在第5、6、8章中讨论，而消息传递型和非一致性共享地址空间系统在第7章中讨论。我们根据对应的程序设计模型编写了关于这些抽象的程序。因为这两种模型的程序的编写有很大不同（如第2章所述），而且某些重要的特征也是不同的，我们采用为一致性共享地址空间所编写的程序来说 243
明工作负载的特征化。特别要指出的是，我们使用了6个以批处理模式运行（即一次运行一个）的不包含操作系统行为的并行应用和计算内核以及一个确实包含了操作系统行为的多道程序工作负荷。尽管我们使用的工作负载的数量不多，但这些应用代表了重要的计算类型，具有变化范围大的特征。

4.4.1 工作负载案例分析

我们用于共享地址空间体系结构的所有并行程序取自SPLASH-2应用集（见附录）。在前面的章节中，作为实例研究我们已经描述和使用了其中的3个（Ocean，Barnes-Hut and Raytrace）。这一节将简要地介绍一下我们将要使用但尚未讨论过的工作负载：LU、Radix、Radiosity和Multiprog。LU和Radix是计算内核；Radiosity是真正的应用；而Multiprog是一个多道程序工作负载。在4.4.2节中，我们将测量这些工作负载与方法有关的某些执行特征，包括数据访问的细目分类、通信与计算的比及其扩放、重要工作集的尺寸和扩放。我们使用这种特征化来为这些应用和后续几章的数据集合选择存储器系统的参数。

1. LU

密集LU因子分解是将一个密集矩阵A转换成为两个矩阵L和U的过程，L和U分别是下三角矩阵和上三角矩阵，它们的乘积等于A（即$A = LU$）[⊖]。它用于解线性方程组，在科学计算类应用以及像线性规划这样的优化方法中会碰到它。它是一个结构良好的内核，虽然

⊖ 如果一个矩阵的大部分元素非零，我们称其为密集矩阵（大多数元素都为零的矩阵叫做稀疏矩阵）。像L这样的下三角矩阵，其主对角线以上的元素均为零，而像U这样的上三角矩阵，其主对角线以下的元素都为零。（主对角线是从矩阵的左上角到右下角的对角线。）

不简单，但是为人们所熟悉且易于理解（Golub and Van Loan 1997）。

LU 因子分解的操作类似于高斯消去法，通过从一行中减去其他行的标量乘积而一次消去一个变量。LU 因子分解的计算复杂度是 $O(n^3)$，而数据集的大小是 $O(n^2)$。我们从第 3 章关于时间局部性的讨论中知道，这是通过分块发掘时间局部性的理想情况。事实上，我们使用分块的 LU 因子分解，不论是在串行还是并行的情况下，都远比非分块版本的效率要高。待因子分解的 $n \times n$ 的矩阵被划分成若干 $B \times B$ 大小的块，其思想是在移向下一块之前
244 应尽可能多地重用一个块。我们现在认为矩阵是由 $n/B \times n/B$ 个块而不是 $n \times n$ 个元素组成，就像我们对元素所做的那样，一次消去和更新一块。此时对于小的 $B \times B$ 的块使用乘和求反这样的矩阵运算，而不是使用作用于元素的标量运算。图 4-11 显示了这种分块的 LU 因子分解的串行伪代码，它也定义了某些相关的术语。

```
for k ← 0 to N-1 do                          /*loop over all diagonal blocks*/
  factorize block A[k,k];
  for j ← k+1 to N-1 do                      /*for all blocks in the row of, and
                                               to the right of, this diagonal block*/
    A[k,j] ← A[k,j] * (A[k,k])^-1;           /*divide by diagonal block*/
    for i ← k+1 to N-1 do                    /*for all rows below this diagonal block*/
      for j ← k+1 to N-1 do                  /*for all blocks in the corresponding row*/
        A[i,j] ← A[i,j] - A[i,k]* (A[k,j])^T;
      endfor
    endfor
  endfor
endfor
```

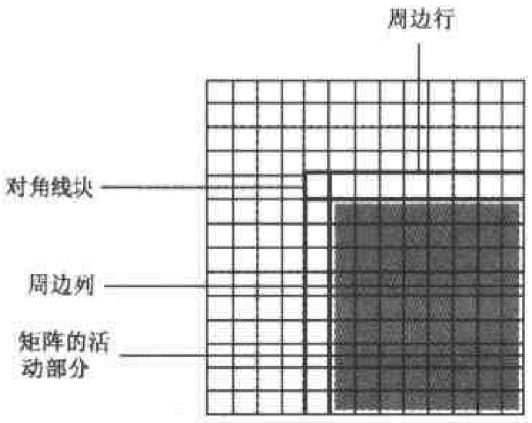


图 4-11　描述串行分块的密集 LU 因子分解的伪代码。N 是各维的块数（$N = n/B$），我们把矩阵看作 $N \times N$ 的块矩阵而不是 $n \times n$ 的元素矩阵。那么，$A_{i,j}$ 代表矩阵 A 的第 i 行和第 j 列的块。在最外层循环的第 k 次迭代中，我们称在 A 的主对角线上的块 $A_{k,k}$，k 为对角块，块的第 k 行和第 k 列分别为周边行和周边列。注意第 k 次迭代并不涉及矩阵前 $k-1$ 行和 $k-1$ 列中的任何块，也就是说，在当前最外层循环中，只有那些位于对角块右下方的正方形区域中的矩阵阴影部分是“活动”的。矩阵的其余部分已经在前面的迭代中计算过了，并且在后续的因子分解中也不再活动。在一个非分块的 LU 因子分解中，我们将类似地谈及对角线元素和元素的周边行和周边列

考虑一下分块的优点。如果我们不计算分块，处理器将计算一个元素，然后计算右边下一个分配的元素，如此进行直到当前行的尾，然后处理器将继续下一行。当它回到下一行的第一个活动的元素时，它将重新访问一个周边行元素（该元素在计算前一行的对应活动元素
245 时已经使用过了）。但是，在此之前，计算已经流经了对应于矩阵整个行的数据，如果矩阵很大，那个周边行元素可能已经不再存在于高速缓存之中了。在分块的程序版本中，在图 4-11 的最内层循环各次迭代块层次上的计算中（即行计算 $A_{i,j} \leftarrow A_{i,j} - A^*_{i,k}\ (A_{k,j})^T$），我

们在回到先前访问过的数据之前在一个方向上仅仅前进 B 个元素，而那些先前访问过的元素仍然在高速缓存中而且可以重新使用。针对于各个 $B \times B$ 个块的操作（矩阵乘和因子分解）各自包含了 $O(B^3)$ 的计算和数据访问，每个块元素被访问 B 次。如果我们这样选择块的尺寸 B，使得一个块的 $B \times B$ 或者 B^2 个元素（加上某些其他数据）都能被高速缓存所容纳，那么在给定的块计算中，只有对元素的首次访问不会在高速缓存中命中，后续的访问都会在高速缓存中命中，这样在 B^3 次访问中有 B^2 次扑空，扑空率为 $1/B$。

在并行版本的程序中，我们把更新一个块的计算看作一个任务。图 4-12 提供了在一次最外层循环的迭代中块之间信息流的图示描述，它显示了在并行版的程序中我们如何将块（即任务）分配给处理器。因为计算的本质，朝向矩阵左上部分的块只在计算的最早几次最外层循环迭代中是活动的，而朝向矩阵右下部分的块却与相当得多的工作相关。所以把连续的行或块构成的正方形分配给进程（矩阵的简单的域分解）会导致不好的负载平衡。因此，我们在两个维度上让块在进程间交错，产生了一种叫做矩阵二维散布分解的划分：可以认为进程构成了$\sqrt{p} \times \sqrt{p}$的网格，这个进程的网格就像糕饼切割器一样反复地压印在块矩阵上。进程负责对以这种方式分配给它的块进行计算，只有它能对这些块写入。这种交错缓解但没消除负载的不平衡，而分块保持了局部性也允许我们在消息传递型的系统中使用较大的数据传输。

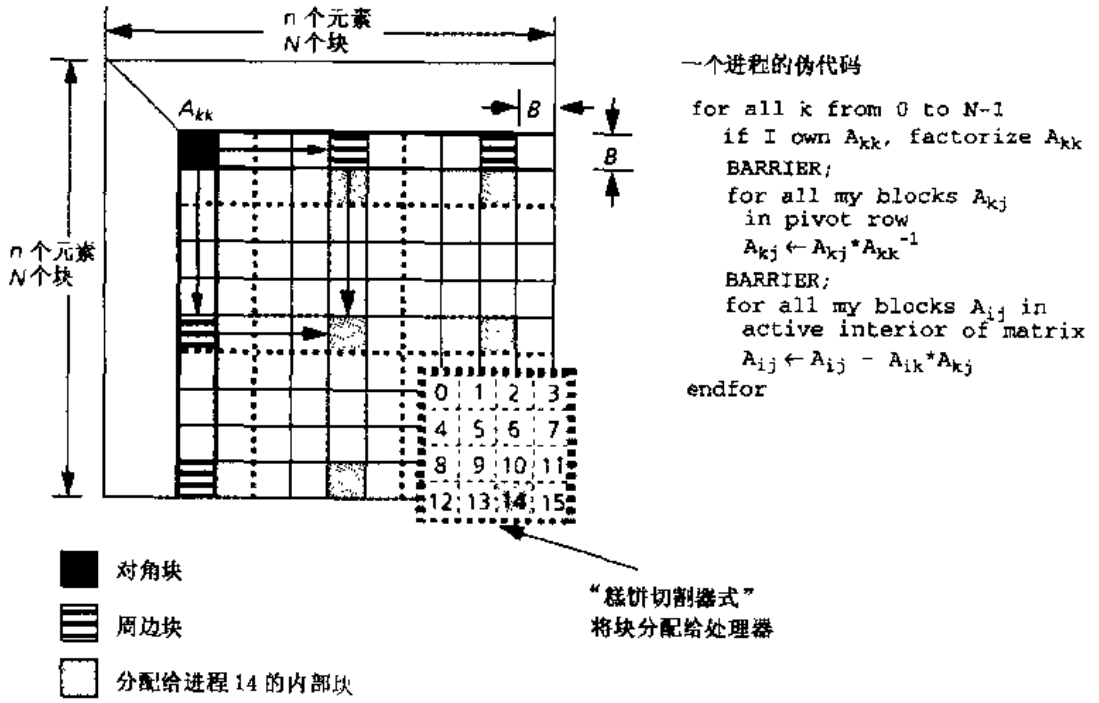


图 4-12　并行的分块式 LU 因子分解：信息的流、划分、并行伪代码。实线箭头显示了在一次外层（k）循环迭代中的信息流。在第一阶段，信息（数据）从对角块（它首先因子分解）流向周边行中的所有块。在第二阶段，矩阵活动部分的块需要来自周边行和周边列的对应元素

把计算分解成块而不是单个元素的缺点在于加大了任务的粒度并损害负载的平衡：因为块的数量比元素少，并发性降低了，每次迭代最大的负载失衡是与一个块而不是单个元素有关。在串行 LU 因子分解中，对块尺寸的惟一约束是：在一次块计算中用到的两个或三个块能容纳于高速缓存。在并行的情况下，理想的块尺寸 B 是由数据局部性和通信额外开销

（特别是在消息传递型的机器上）之间的折中而决定的，要降低通信开销倾向于较大的块；而另一方面，负载平衡则倾向于较小的块。所以理想的块尺寸取决于问题的规模、处理器的数量和其他体系结构参数。在实践中，16×16 或 32×32 个元素的块尺寸在大型的并行机器上工作得很好。

分块提供了块计算中数据的重用，数据也能跨越不同的块的计算而被重用。为了重用来自远地块的数据，我们可以显式地在主存中复制块并保持它们，或者在高速缓存一致性的机器上，依赖于足够大的高速缓存自动地完成这一点。但是，对改善性能而言，跨越块计算的重用不像块内的重用那样重要（显式的复制有其代价），所以在我们的程序中，我们并不在主存中显式地复制块。

至于空间局部性，因为现在分解的单位是二维的块，问题与 3.3.1 节讨论简单方程求解器内核时的问题很类似。所以我们用一个四维数组的数据结构来表示共享地址空间的矩阵，使得一块中的数据在地址空间中连续。前两个维度说明一个块，后两个维度说明块中的一个元素。这种表示方法允许我们以页的粒度在存储器中适当地分布块。（如果块比页小，我们可以用另外一个额外的数组维度来保证分配给一个进程的所有块在地址空间内是连续的。）然而，分块使得容量型扑空率足够的小，从而使 LU 因子分解时主存中的数据分布不成为主要的问题。使用高维数组保持一个块的数据连续的更重要的理由是，减少跨越一个块的子行和跨越块时的高速缓存的映射冲突，我们将在 5.6 节讨论这个问题。高速缓存冲突对于数组的尺寸和处理器的数量非常敏感，特别是对于直接映射的第一级高速缓存更是如此，它很容易抵消掉分块所带来的大部分好处。

246
↓
247

并行 LU 因子分解没有使用加锁操作。使用了栅障操作分隔最外层循环迭代以及迭代中的不同阶段（例如，保证在使用一个周边行中的块之前先计算该行。）可以使用块级别的点对点同步来发掘更高的并发性，但是使用栅障使得编程容易得多。

2. Radix

程序 Radix 使用流行的基数排序法对一系列被称为键字的整数排序。假定有 n 个整数需要排序，每个整数的长度为 b 比特。该算法使用 r 个比特的基数，这里 r 由用户选择。这意味着一个 b 比特的键字可以被表示为一个包含 $\lceil b/r \rceil$ 个组的集合，每个组为 r 个比特（见图 4-13）。该算法经历 $\lceil b/r \rceil$ 个阶段或迭代。每个阶段从最低位的组开始，根据键字在所对应的 r 比特组中的值对键字排序，r 比特的组叫做一个数位[○]。在 $\lceil b/r \rceil$ 个阶段的末尾，键字被完全排序。在每个阶段中使用两个一维的大小为 n 的整数数组：其中一个叫做输入数组，存储该阶段输入的键字；另一个叫做输出数组，存储从该阶段输出的键字。一个阶段的输入数组是下一个阶段的输出数组，反之亦然。

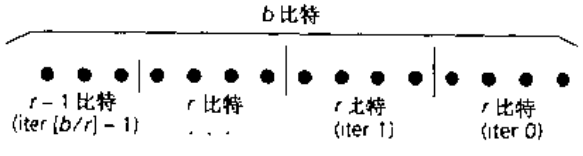


图 4-13　一个 b 比特的数字（键字）被分成 $\lceil b/r \rceil$ 个组，每组 r 个比特。radix 排序的首次迭代使用最低有效的 r 比特，依次类推

考虑在一个阶段内部的并行计算，它根据键字在某一特定数位上的值对所有的键字排序。并行算法把每个数组中 n 个键字对 p 个进程做划分，将前 n/p 个键字分配给进程 0，接

○ 从最低位而不是最高位的 r 比特组开始排序的原因在于它能导致一个“稳定”的排序，也就是说，具有相同值的键字在输出中的相对位置与它们在输入中的相对位置一样。

下来的 n/p 个键字分配给进程 1，依次类推。分配给一个进程的那部分数组被放入对应的处理器的本地存储器中。在某个阶段分配给一个进程的输入数组中的 n/p 个键字叫做该阶段该进程的本地键字。在一个阶段中，进程执行下列步骤：

1）扫描 n/p 个本地键字，建立一个键字值的局部（每进程一个）直方图。该直方图有 2^r 个柱方项，这里 r 是数位中的比特数。如果碰到的键字在当前阶段的值为 i，那么直方图的第 i 个柱方项就加 1。

2）当所有的进程都完成了步骤 1）（由程序中的栅障同步决定）后，把局部直方图累计到一个全局的直方图去。如习题 4.14 所讨论的那样，这一操作通过一个并行的前序计算完成。全局直方图记录了对于当前数位的每个取值各存在多少个键字，还记录了对于每一个进程 ID 值 j，有多少个具有给定值的键字被 ID 值小于 j 的进程所拥有。

3）对 n/p 个本地键字做另一次扫描。对于每个键字，使用局部和全局直方图决定应该 248
把该键字放到输出数组的哪一个（已排序）的部分去，并将键字的值写入输出数组的项中去。注意，要写入的数组项很可能是非本地的，这种可能性的期望值是 $(p-1)/p$（见图 4-14）。这个步骤叫做置换步骤。

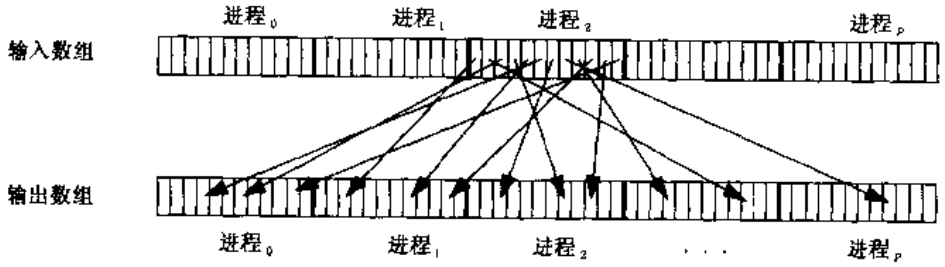


图 4-14 radix 排序阶段的置换步骤。在每个输入和输出数组中（它在后续的阶段中改变位置），分配给一个进程的键字（数组项）被分配到对应处理器的本地存储器中

读者可以在文献（Blelloch et al. 1991；Culler et al. 1993）中找到对 radix 排序算法及其实现的更详细的说明。在共享地址空间型的实现中，通信发生在置换阶段写入键字的时刻（或者，如果键字停留在写入者的高速缓存中，发生在下一次迭代的直方图生成阶段读出键字的时刻），也发生于从局部直方图构造全局直方图的时刻。与置换相关的通信是全部对全部的个人化通信（即每个进程与各个其他进程交换它的键字的不重叠的子集），但是，它是不规律和散布的，其精确的模式依赖于键字的分布。同步包括阶段之间的全局栅障和阶段内建立全局直方图的较细粒度的同步。后者可以采取互斥或者点对点的事件同步的形式，这取决于该阶段的实现（见习题 4.14）。

3．Radiosity

Radiosity 方法用于计算机图形学，用来计算包含散射表面的场景的全局照度。在层次型的 radiosity 方法中，一个场景最初被建模成包含了 k 个大的输入多边形，或者说片。例如， 249
桌面或椅背可以是一个输入的片。我们要计算这些片中每对片之间光的传递相互作用。可以简单地认为这个算法完成以下功能：如果一对片之间光的传递比阈值强，其中之一（比如说大的那个）将被进一步分割，在由此产生的子片和其他片之间递归地计算光的相互作用。这个过程一直持续到所有片对之间的光传递都很低为止。所以，为了改善照度计算的精度，根据需要将片层次式地分割成子片。每一次分割产生 4 个子片，使每个片形成一个四叉树。如果最后不再分割的子片的数量是 n，那么对初始片数为 k 的情况，算法的计算复杂性是

$O(n+k^2)$。下面给出算法步骤的简单说明。读者能够在文献（Hanrahan，Salzman，and Aupperle 1991；Singh 1993）中找到算法的细节。

构成场景的输入片首先被插入一个二进制空间分割（BSP）树（Fuchs，Abram，and Grant 1983），它是一个便于片对之间可见性的高效计算的数据结构。初始时给每个输入片设立一个相互作用链表，该链表记录从该片能够看到的输入片，对这些片它必须计算相互作用。然后，通过下面的迭代算法计算照度：

1）对每一个输入片，计算由它的相互作用链表中所有的其他片而形成的它的照度，如果需要，将它或者其他片层次式地分割，递归地计算它们的相互作用（见图4-15）。

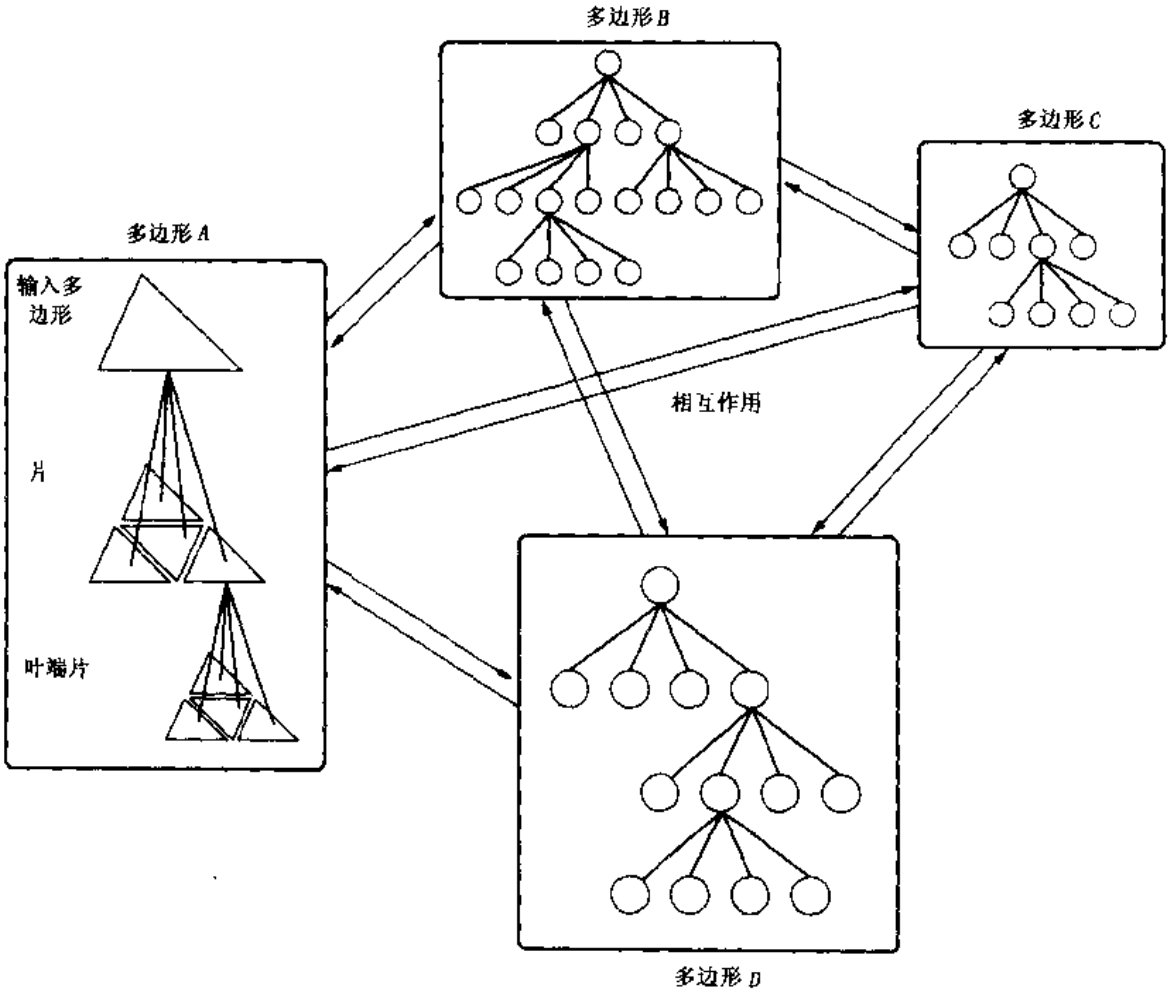


图4-15　随着照度计算的进展将输入多边形层次式地分割成四叉树。每一个输入多边形产生一个片的四叉树，该四叉树与其他四叉树的片相互作用

2）从位于四叉树的叶片处的片开始，把所有片的照度相加（由它们的面积加权）获得场景的总照度，将它与前一次迭代形成的照度比较，检查是否在确定的容差范围内收敛。如果照度未收敛，返回步骤1），否则的话，转向步骤3）。

3）为了显示对结果做光滑处理。

一次迭代的大部分时间花费在步骤1），因此让我们对它进一步考察。假定一个片 i 正在遍历其相互作用链表以计算它与其他片（四叉树节点）的相互作用。与另一个片（比如说 j）的相互作用包含这两个片的相互可见度计算以及它们之间的光的传递。（实际的光传递

是实际相互可见度与如果没有遮蔽因而相互完全可见而产生的光传递的乘积。）计算相互可见度包含从一个片到其他片遍历几次 BSP 树[⊖]；事实上，可见度计算占了整个执行时间的非常大的部分。如果相互作用的结果指出“源”片 i 应该被进一步分割，那么就对片 i 生成四个孩子，如果它们并没有因为先前的相互作用而已经存在；从片 i 的相互作用链表中去掉片 j 并将 j 加到 i 的每一个孩子的相互作用链表中去，这样这些相互作用会在以后计算。如果结果指出片 j 必须被进一步分割，那么片 j 在片 i 的相互作用链表中被 j 的孩子所替代。这意味着在片 i 和片 j 的每一个孩子之间将计算相互作用。这些相互作用本身可能又会引起进一步的分割，从而使得这个过程递归地继续下去（即如果片 j 的孩子在计算相互作用的过程中被再划分，片 i 最终要与片 j 之下的一棵片形成的树计算相互作用）。因为从一次分割产生的一个片的四个孩子在片 i 的相互作用链表中原地替换了其父亲，对由片 j 的后代组成的树的遍历是深度优先的。片 i 的相互作用链表被完全遍历之后才转向处理下一个片的相互作用链表（下一片可能是片 i 的后代或是另一个不同的片）。图 4-16 显示了这种层次式的相互作

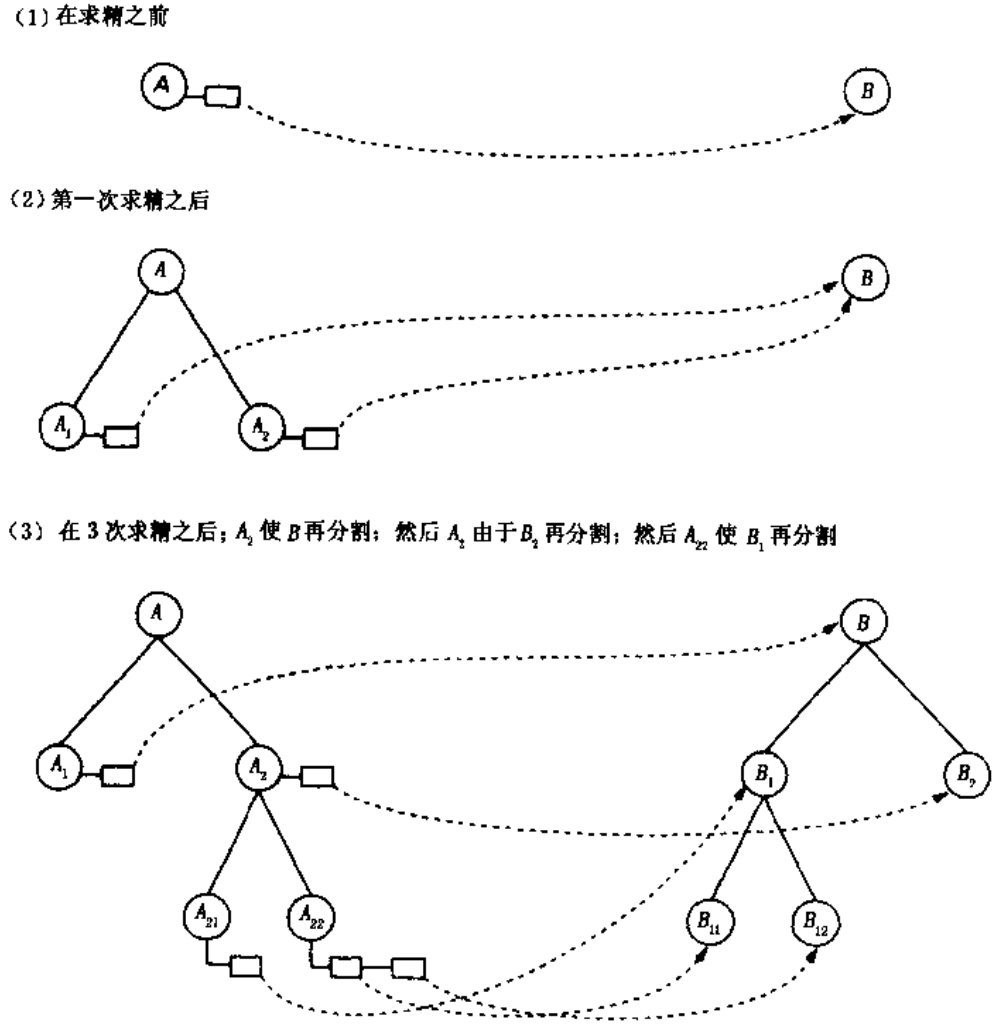


图 4-16　相互作用和相互作用链表的层次式求精。为了清楚起见，这里显示的是二叉树而不是四叉树，而且只显示了一个输入多边形的相互作用链表

⊖ 可见度的计算是在两个片之间概念性地发射一些射线并观察有多少射线未被场景中处于其间的其他干涉片遮蔽而到达目的地片。对每根这样的概念性射线，决定它是否被遮蔽的工作可以通过遍历从源到目的地片的 BSP 树而有效地完成。

用的不断求精过程。一轮迭代计算之后，所有的相互作用和求精都被完成，迭代算法的下一次迭代又以上一次迭代结尾形成的四叉树和相互作用链表开始。

在这个应用中可以找到三个层次的并行性：跨 k 个输入多边形，跨再分割这些多边形而形成的片，跨对一个片计算的相互作用。所有这三个层次都包含处理器之间的通信和同步。根据问题的规模、处理器的数量和机器的特性，我们把一个任务定义为一个片及其所有相互作用或者单个片对片的相互作用，从而获得最佳的性能。

因为计算和分割高度不可预见，我们必须使用任务队列和任务窃取来平衡工作负载。并行化的实现为每个处理器提供了它自己的任务队列。一个处理器的任务队列被初始化为初始可用的多边形-多边形相互作用的一个子集。当由于相互作用而分割一个片时，为这些子片而建立的新的任务被排入计算了相互作用并进行再分割的处理器的任务队列中。处理器从它的队列执行任务直到队列中没有任务为止。然后，它从其他处理器的队列中窃取任务执行。任务队列由加锁操作保护，当片被分割时，加锁提供了对片的互斥访问。（注意，分配给两个不同进程的两个片可能在它们的相互作用链表中有相同的片，所以两个进程可能会试图同时分割该片。）在一次迭代的两个步骤之间使用栅障同步。由于任务窃取和相互作用及再分割计算的次序，并行算法是非确定性的，它具有高度非结构化和不可预见的通信及数据访问的模式。

4．Multiprog

到目前为止我们讨论的工作负载仅仅包括一次运行一个程序的并行应用。但是，多处理器的通常用法，特别是小规模的共享地址空间多处理器，是作为多道程序工作负载的吞吐引擎。这类机器所支持的细粒度资源共享允许单一操作系统映像有效地服务于多个处理器。操作系统的活动通常是这样的工作负载的重要组成部分，而操作系统本身构成了重要的、复杂的并行应用。最后研究的一个工作负载是一个多道程序的（时分的）工作负载，由一系列串行应用和操作系统本身所组成。应用是两个 UNIX 文件压缩作业和两个并行编译，或者说 pmakes，在并行编译应用中，生成执行文件所需的多个文件被并行地编译和汇编。操作系统是由 Silicon Graphics 生产的 UNIX 版本，叫做 IRIX（版 5.2）。

4.4.2　工作负载的特征化

现在我们对所有我们提到过的工作负载的某些基本的特征定量化，包括被区分成读和写或共享和私有的数据访问、并发性和负载平衡、固有的通信与计算的比率及其扩放的方式、重要工作集的尺寸和扩放。与空间局部性相关的特征在后续章节中谈到特定的体系结构风格时再予以测量。在本节中，将给出并行应用在 16 个处理器上执行的定量特征数据和多道程序在 8 个处理器上执行的定量特征数据。我们还定性地或解析地讨论所涉及的特征如何随问题的规模而扩放，某些时候对它们进行测量。

250～253

1．数据访问和同步特征

表 4-1 总结了不同的工作负荷中基本的访问次数和同步事件（加锁和全局栅障）的动态频度。除非特别说明，输入数据集合符合本书中一直使用的缺省问题的尺寸。所选择的问题的规模足够大，可以对多至 64 个处理器的机器做实际评价；但又足够小，可以在合理的时间内被模拟。所以它们是处于在 64 处理器的机器上实际运行的数据集的小尺寸的一端，但是也很适合于较小规模的系统。

表 4-1　关于应用程序的综合统计数据

应用	输入数据集	指令总数（M）	总的 FLOPS（M）	访问总数（M）	读出总数（M）	写入总数（M）	共享读（M）	共享写（M）	栅障	加锁
LU	512 × 512 矩阵 16 × 16 块	489.52	92.20	151.07	103.09	47.99	92.79	44.74	66	0
Ocean	258 × 258 网格 容差 = 10^{-7} 4 个时间步	376.51	101.54	99.70	81.16	18.54	76.95	16.97	364	1 296
Barnes-Hut	16K 粒子 θ = 1.0 3 个时间步	2 002.74	239.24	720.13	406.84	313.29	225.04	93.23	7	34 516
Radix	256K 整数 radix = 1 024	14.02	—	5.27	2.90	2.37	1.34	0.81	11	16
Raytrace	汽车场景	833.35	—	290.35	210.03	80.31	161.10	22.35	0	94 456
Radiosity	室内场景	2 297.19	—	769.56	486.84	282.72	249.67	21.88	10	210 485
Multiprog 用户	SGI IRIX 5.2 两个 pmakes +	1 296.43	—	500.22	350.42	149.80	—	—	—	—
Multiprog 内核	两个 compress	668.10	—	212.58	178.14	34.44	—	—	—	621 505

注：对于并行程序，共享的读和写不过是指由应用进程发出的所有非堆栈的访问。所有这样的访问并不需要指向真正由多个进程共享的数据。工作负载 Multiprog 不是并行应用，所以它并不访问共享数据。表项中的短横线代表该项测量不适用或者对该应用没有做测量（例如，Radix 没有浮点数操作）。（M）表示在该行的测量是以百万为单位的

我们只是在父节点生成了子节点之后记录行为和定时统计数据。先前的访问（由主进程发出的）被模拟但并没有包括在统计数据之中。在大多数应用中，精确的测量在子进程被创建之后开始。但 Ocean 和 Barnes-Hut 两个程序是例外。在这两种情况下，我们能够利用机会，大大减少模拟所需要的时间步数（如 4.3.2 节所讨论的那样）；但是，我们必须忽略冷启动的扑空并允许应用在开始测量之前稳定下来。我们模拟很少几个时间步（对 Ocean 来说是 6 步，对 Barnes-Hut 来说是 5 步），在最初两个时间步之后就开始记录行为和定时统计数据。对于工作负载 Multiprog 来说，从靠近 pmake 开始位置的检查点收集统计数据，而对于所有其他的应用，仅仅考虑应用的数据访问；但对 Multiprog，还考虑指令访问的影响，进一步把内核的访问和用户应用的访问划分成独立的类。该表显示了不同的工作负载在把操作分为整数操作和浮点数操作、读和写、共享和私有等方面有很大的差别，表明它们很好地覆盖了这些坐标。

2．*并发性和负载平衡*

负载平衡的特征化是通过测量算法的加速比，也就是说，测量在 PRAM 体系结构模型上的加速比（第 3 章所讨论的）完成的，它假定数据访问和通信具有零时延（它们仅花费发出访问的指令所需的时间）。相对于理想加速比的偏离归咎于负载的不平衡、临界区的串行性以及由冗余计算和并行性管理所产生的额外的工作。

图 4-17 显示了使用缺省的数据集合，在多达 64 个处理器的系统上 6 个并行程序的算法加速比。其中 3 个程序（Barns-Hut、Ocean 和较小范围的 Raytrace）即使使用相对较小的数据集，在多达 64 个处理器的情况下仍有良好的加速比。这些程序的主要阶段是在大的数据集合（Barnes-Hut 中所有的粒子、Ocean 中的整个网格和 Raytrace 中的图像像素）上的数据并行。它们只是在某些全局归约操作和某些特殊阶段中受到有限的并行性和串行化的影响，而

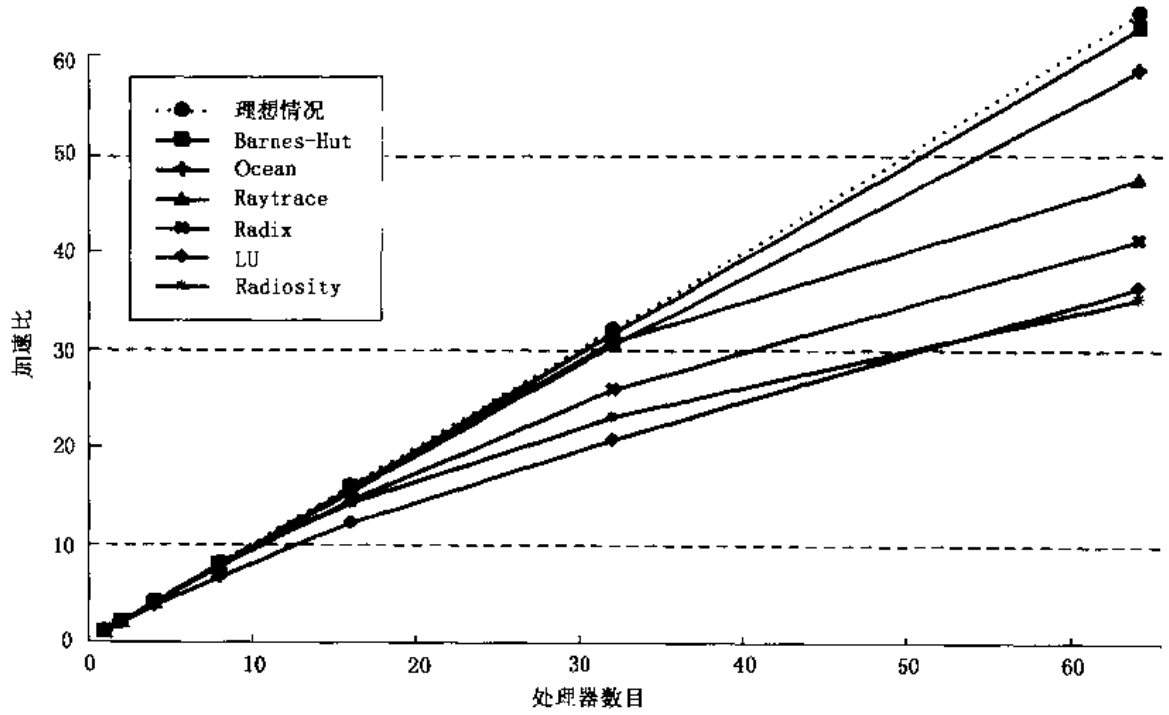


图 4-17　6个并行应用的算法加速比。理想的加速比曲线表示在 p 个处理器情况下加速比为 p

那些特殊阶段就所执行的指令数而言并不占主导地位（例如，Barnes 中树的构造和靠近根的向上扫描的部分，Ocean 的多网格层次结构的较高的层次）。Raytrace 有一个产生麻烦的高度竞争的临界区，引起了串行化（事实上，这个临界区对于正确的执行并不一定是需要的，是为了记录某些重要的统计数据而使用它）。

所有 6 个程序在多达 16 至 32 个处理器的情况下都表现了良好的加速比。在处理器数量较多的情况下使用这些数据集加速比不是非常好的程序有 LU、Radiosity 和 Radix。在这几种情况下，其原因是输入数据集的尺寸而不是应用所固有的负载非平衡的特性。在 LU 中，尽管采用了面向块的分解，缺省的数据集导致在 64 个处理器的情况下负载显著的不平衡。较大的数据集（或较少的处理器）通过在分解的每一个步骤给每个处理器提供更多的块而降低不平衡性。对于 Radiosity，不平衡性也是由于使用了较小的数据集，虽然分析起来非常困难。最后，对于 Radix 在 64 个处理器时，较差的加速比是由将局部直方图累加到全局直方图
256 时的前序计算所致（见 4.4.1 节），它不能被完全并行化。在这个前序计算上所花费的时间是 $O(\log p)$，而在其他阶段所花费的时间是 $O(n/p)$，因此当排序的键字的数量增加时，不平衡的阶段在整个工作中所占的比重将下降。所以，当选择较大的数据集时，即使是这 3 个程序也能被用来评价规模较大的机器。

我们已经满足了标准，这就是不要选择那些本质上不适合于所希望评价的机器规模的并行程序，并要懂得如何为现有规模机器上的程序选择合适的数据集。现在来考察一下固有的通信与计算的比以及程序的工作集尺寸。

3. 通信与计算的比

在通信与计算的比中，我们包括了固有的通信和一个字第一次被处理器访问时所产生的通信（即在测量启动之后发生的冷启动扑空）。在可能的条件下，数据是在物理分布的存储

器内适当地分布的，所以能够认为这种冷启动通信是本质性的而不是附加造成的。为了避免由于容量的限制或空间局部性不好所造成的附加的通信，我们模拟了每个处理器有无限多的高速缓存和每个高速缓存块由单个字组成的情况。我们所测量的通信与计算的比是所有的处理器平均来看，每条指令所通信的应用数据的字节数。对于浮点数密集型的应用（LU 和 Ocean），我们是使用每个浮点数操作（FLOP）所通信的字节而不是每条指令所通信的字节，因为 FLOP 的数量与指令的总数相比，对编译器的行为不那么敏感。

我们将首先观察一下对于表 4-1 所示的基本问题规模所测量到的通信与计算的比和所使用的处理器的数量的关系。这显示了在问题规模不变的前提下，该比值如何随处理器的数量而上升。然后，在任何可能的情况下，我们将解析地考察该比值是如何依赖于数据集的尺寸和处理器的数量的（表 4-2）。其他应用参数对通信与计算的比的影响将单独讨论，并且通常只是定性地讨论。

表 4-2 固有的通信与计算比的增长速率

应 用	增长速率
LU	$\sqrt{P}/\sqrt{DS}$
Ocean	$\sqrt{P}/\sqrt{DS}$
Barnes-Hut	大约$\sqrt{P}/\sqrt{DS}$
Radiosity	不可预见
Radix	$(P-1)/P$
Raytrace	不可预见

注：DS 是数据集的尺寸（以字节为单位），P 是处理器的数量。

图 4-18 给出了在基本的问题规模条件下，对我们的 6 个并行程序的测量结果。我们注意到的第一件事是平均的固有的通信与计算的比通常相当小。对于以 400 MIPS（每秒的百万指令）的速度工作的处理器来说，每条指令 0.1 字节的比率意味着大约 40 MBps 的数据流量，对于现代高性能的多处理器网络而言，这是相当小的。实际的流量比固有的流量要高得多，这既是由于附加造成的通信，又是由于在每次传输中都要有控制信息与数据一起传送。它指出，正是通信的突发性、通信的其他来源和通信的模式（比如，全部对全部或者长距离的通信）可能会引起通信带宽的问题。平均比率相当高的惟一的应用是 Radix，所以对于这个应用，在评价中仔细地模拟通信带宽就特别的重要。通信与计算的比低的一个原因在于，257
我们所使用的这些应用在被安排并行执行时已经很好地优化了。在实际中使用的应用，包括这些应用的其他版本，则可能呈现较高的通信与计算的比。

从该图我们观察到的下一个事实是不同的应用的通信与计算比的增长速率很不相同，显示了对这一行为特性有着的良好覆盖。表 4-2 以解析的方式总结了增长速率与处理器数量和数据集尺寸（未在图中显示）的关系。如果我们在某些应用（比如 Ocean）中使用不同的数据集的尺寸，通信与计算比将会有巨大的变化；但是在其他应用中，至少固有的通信不会有多大的变化。附加产生的通信则完全不同，在后面的章节中，我们将根据不同的体系结构的情况，考察由附加产生的通信所造成的通信流量。

显然，尽管增长速率是一个基本的参数，但重要的是应该认识到它并未揭示通信与计算比的表达式中的常数因子，在实践中，该因子比渐进线式的增长速率更为重要。例如，如果一个程序的通信与计算比仅随处理器的数量而对数式地增长，那么，以渐近线的增长的渐进 258

线方式看，它确实要比其比率按处理器数量平方根而增长的应用要小；但是，如果它的常数大得多的话，对于所有实际的机器规模，它的比率实际上要大得多。从图 4-18 中能够决定应用的常数因子。

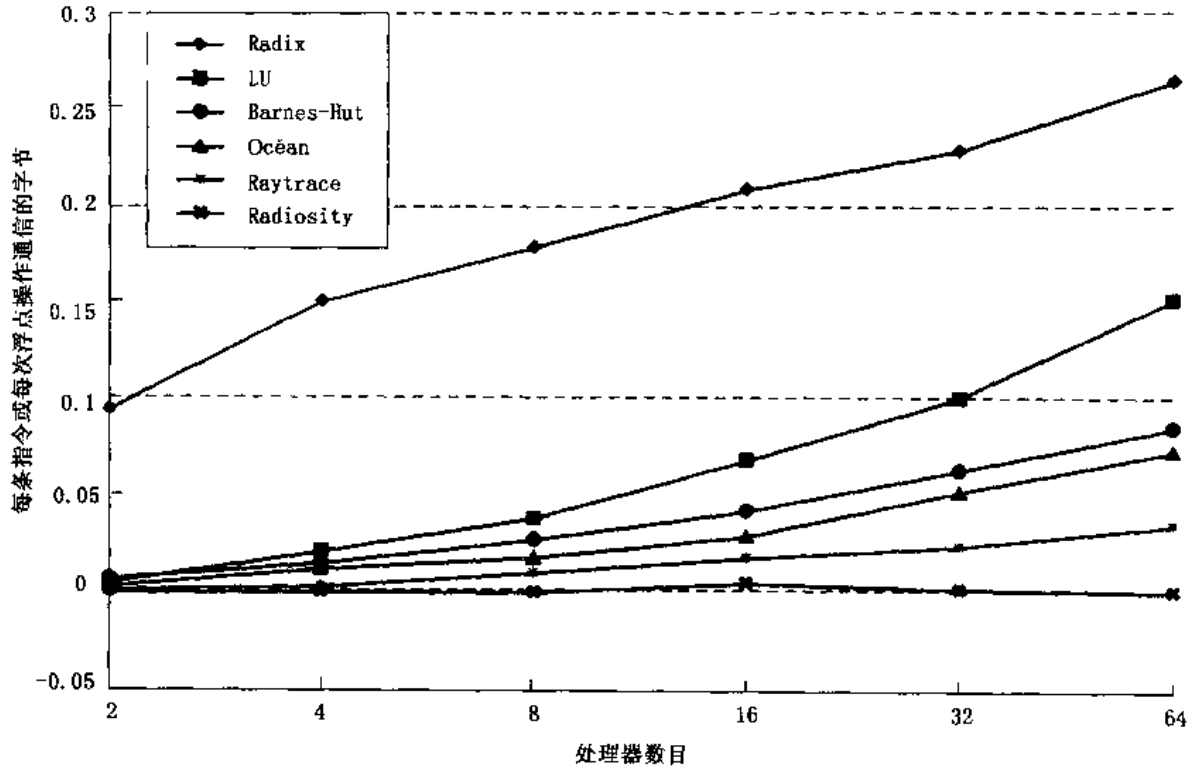


图 4-18　在 6 个并行程序中，对于基本的问题规模，通信与计算之比与处理器数量的对照

4．工作集的尺寸

测量一个程序的固有工作集的尺寸的最好办法是采用全相联的高速缓存和大小为一个字的高速缓存块，以不同尺寸的高速缓存模拟该程序的运行，发现扑空率和高速缓存尺寸的对照曲线的拐点。较小的相联度可能会使保持工作集所需要的高速缓存的尺寸大于固有的工作集的尺寸，使用多字组成的高速缓存块也会产生这一效果（由于高速缓存中的碎片所致）。在测量中，通过使每个处理器具有单级全相联高速缓存，采用最近最少使用（LRU）的高速缓存替换策略和 8 字节的高速缓存块，测量很接近固有工作集的尺寸。一般使用 2 的幂作为高速缓存的尺寸，但是为了识别拐点，在扑空率随高速缓存尺寸的变化更为显著的情况下，我们以更小的粒度改变高速缓存的尺寸。

图 4-19 显示了 6 个并行应用的扑空率与高速缓存尺寸的对照曲线，工作集被标注为 1 级工作集（L_1 WS）、2 级工作级（L_2 WS）等等。如我们在第 3 章所讨论的那样，像 Ocean 这样的应用有几个工作集，但我们的注意力集中在两个定义最清楚和最重要的工作集上。除了这些工作集以外，某些应用还有微小的、其尺寸不随问题规模和处理器数量而扩放的工作集；因此它们总是可以被容纳于最靠近处理器的高速缓存中，称这些工作集为 0 级工作集（L_0 WS）。典型地，这种工作集是由堆栈数据所组成，这种堆栈数据被程序作为对某一特定基本计算（例如 Barnes-Hut 中的粒子元的相互作用）的临时存储而使用并且在这些计算中反复使用。当这些工作集可以被观察到时，将它们在图中标出，但是不对它们作进一步的讨论。

259

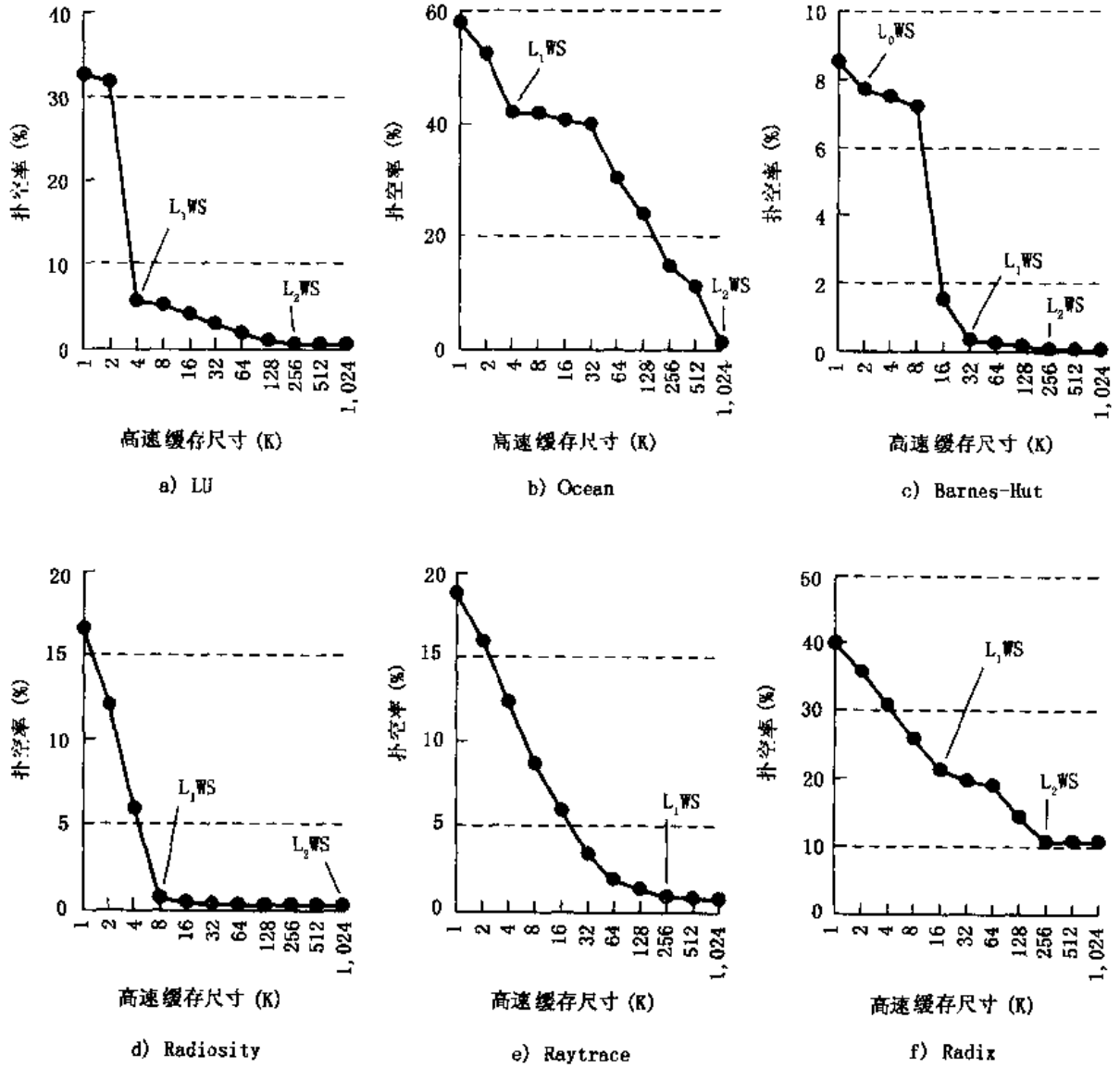


图 4-19　在 16 个处理器执行的情况下 6 个并行应用的工作集曲线。这些图显示了在每个处理器具有全相联第一级高速缓存和 8 字节的高速缓存块的情况下，扑空率和高速缓存尺寸的对照

我们看到在大多数情况下，工作集是非常明确地定义的。表 4-3 总结了对于不同的工作集其尺寸如何随应用参数和处理器数量而扩放；对于性能而言它们是否重要（至少在高效的高速缓存一致的机器上）以及对于合理的问题尺寸，是否能期望这样的工作集无法被现代的 260
第二级高速缓存（具有合理的高速缓存相联度，至少高于直接映射）所容纳。其工作集在最后两列都为“是”的应用是 Ocean、Radix 和 Raytrace。回忆一下，在 Ocean 中，所有主要的计算流过一个或多个网格在一个进程中的分区。大的工作集包含整个网格在一个进程中的分区，它通过重用而受益。所以，这个大工作集是否能为现代的第二级高速缓存所容纳取决于网格的尺寸和处理器的数量。在 Radix 中，一个进程流过它的所有 n/p 个键字，同时大量地访问直方图数据结构（与所用的基数成比例）。所以，让高速缓存容纳直方图是重要的，但这个工作集没有被明确地定义，因为键字也同时流过。较大的工作集包含键字数据集在进程的整个划分，它或许能或许不能被高速缓存所容纳，这取决于参数 n 和 p。最后，在 Raytrace

表 4-3　SPLASH-2 测试程序集的重要工作集和它们的增长速率

程序	工作集 1	增长速率	重要?	实际中无法容纳于高速缓存?	工作集 2	增长速率	重要吗	实际中是否无法容纳于高速缓存
LU	一块	固定的(B)	是	否	*DS* 的划分	*DS*/*P*	否	是
Ocean	少数子行	$\sqrt{P}/\sqrt{DS}$	是	否	*DS* 的划分	*DS*/*P*	是	是
Barnes-Hut	一个星体的树状数据	$(\log DS)/\theta^2$	是	否	*DS* 的划分	*DS*/*P*	否	是
Radiosity	BSP 树	log(*polygons*)	是	否	非结构化	非结构化	否	是
Radix	直方图	Radix *r*	是	否	*DS* 的划分	*DS*/*P*	是	是
Raytrace	在光线之间重用的场景和网格数据	非结构化	是	是	—	—	—	--

注：*DS* 代表数据集的尺寸，*P* 是处理器的数量。

中，已经看到了工作集发散因而是不良定义的，所以可能变得很大，它取决于正在被跟踪的场景的特征和视点。对于其他的应用，我们期望对于实际的问题和机器的规模，重要工作集能够被高速缓存所容纳。在后续章节评价体系结构折中时，根据我们的方法，应该考虑这一点。特别是对于 Ocean、Radix 和 Raytrace，应该选择较大的工作集，覆盖能与不能被高速缓
261 存容纳的两种场景；因为在实践中，这两种情况都存在。

4.5　结论

至此，对多处理器工作负载驱动式评价的主要问题已经有了很好的理解：选择工作负载，扩放问题和机器的规模，处理大的参数空间，选择度量指标。对每一个问题，都给出了一组应遵循的指导方针和步骤、对如何避免误区的理解以及对研究的局限性的理解。我们已经具备了在阅读本书的剩余部分时，做出对体系结构折中的定量说明的基础。本书中的实验是为了说明要点，而不是为了全面地评价那些折中，因为后者要求范围广泛得多的工作负载和参数变化。

我们已经看到了应该选择那些能够代表宽范围的应用、行为模式和优化级别的工作负载。尽管完整的应用和多道程序的工作负载是不可缺少的，但像微基准测试程序和内核程序这样较简单的工作负载也可以扮演重要的角色。

我们也看到了要实现适当的工作负载驱动式评价，要求既要理解工作负载的相关行为特性，也要理解这些特性和体系结构参数之间的相互作用。虽然这个问题是复杂的，但我们已经考察了处理巨大的参数空间的指导方针，以便对真实的机器以及体系结构上的折中进行评价，并且在仍然能覆盖各种真实情况的前提下对参数空间进行剪裁。

规模扩放的问题进一步强调了理解工作负载的相关性质的重要性，它影响所有重要的特征及相互作用。执行时间和存储器两者都可能成为扩放的约束，而应用通常有着一个以上决定关键执行特性的参数。我们基于对这些参数、它们的相互作用、它们对执行时间和存储器的需求的理解而扩放程序的规模。我们看到，由应用需求所驱动的实际的规模扩放模型在评价体系结构重要性方面得出的结论与仅仅扩放单个应用参数的初级模型的结论完全不同。事实上，规模扩放对于设计和评价都是重要的。在评价技术如何适应扩放（比如，相对于存储器和网络速度的处理器速度）的同时，理解应用的扩放以及它的内涵，对于决定未来机器合

适的资源分布是非常重要的（Rothberg，Singh，and Gupta 1993）。

在我们关于选择评价和结果表示的指标的讨论中也有很多有意思的问题。例如，了解执行时间（最好将每个处理器进一步细分为主要组成部件）和加速比都是应该加以表示的很有用的指标，而诸如 MFLOPS 或 MIPS 这样基于速率的指标或利用率指标对特定的目的而言，可能是有用的；但作为通用的指标，它们太容易受到问题的影响。

在本书的后续部分，将举例说明对具有共享地址空间的真实系统及体系结构折中的工作负载驱动式的评价，本章描述了用于该评价的主要工作负载，定量说明了它们的特征。（对于消息传递型的系统，将在第 7 章简要地考察一个标准的消息传递基准测试程序集，即 NAS 并行基准测试程序 II［NPB2］的性能。）现在，我们的基础已经足够牢固，可以进一步研究核心的体系结构和设计。 262

习题

4.1 1）假设为高速缓存一致的机器的通信体系结构建议了一种新的特性，你要对该特性进行评价研究。你的经理告诉你，你在评价中只能使用不超过 3 个并行程序，尽管这不符合你对问题的更好的判断，但你不得不服从。从本章和第 3 章所考察的 7 个并行程序（包括多道程序的工作负载）中，你将选择哪 3 个？为什么？

2）假如你知道该特性是为改善机器的通信带宽而设计的，你的选择会受到什么影响？

3）假如该特性是为了增加用于非本地分配的数据的有效复制存储，你将选择什么程序？

4.2 请说明与 MC 扩放相比较，TC 扩放的根本问题，试举例说明。

4.3 假定你必须评价系统的可扩展性。一种可能的方法是像本章所定义的那样测量不同规模扩放模型下的加速。另一个办法是决定要想达到 70%并行效率，问题的规模应如何扩放。这两个方法的优点，特别是缺点和警示是什么？你实际将如何做？

4.4 你的经理要求你比较两类基于相同的单处理器节点的系统，但它们的通信体系结构有一些有趣的差别。经理告诉你，她仅仅关心 10 个特定的应用。她指示你，在为每个应用指定固定数量的处理器和固定的问题规模（你的选择）的前提下，只需要产生能说明哪个系统更好的单一数字形式的测量；尽管你认为对这些并行应用求平均不是一个好主意。在选定问题的规模之前你还要问她什么额外的问题？你将向她报告哪种平均值的测量？为什么？

4.5 一个系统经常会对某个应用呈现好的加速比，尽管它的通信体系结构对该应用并不是很合适。为什么会发生这样的事？除了加速比之外，你能否设计一种指标，它能够测量通信体系结构对于该应用的有效性。讨论设计这样一个指标可能碰到的问题和不同的方案。

4.6 你读到的一篇研究论文建议了一种通信体系结构机制，并告诉你，以一台具有 32 个处理器的机器的特定通信体系结构为基础，该机制对某些你关心的工作负载的性能改善达 40%。该信息是否足以使你决定将该机制包括到你将要设计的下一台机器中去？如果不是，请列出为什么不这样做的主要理由，并说明你还需要哪些其他的信息。假设你将要设计的机器也具有 32 个处理器。 263

4.7 假设你要设计一些实验来比较在一个共享地址空间机器上实现锁的不同方法，你想要

测量什么样的性能特性？你要设计什么样的微基准程序？你需要的专门性能测量是什么？然后对于全局栅障情况回答相同的问题。

4.8　你已经设计了一个方法，在第一章所讨论的 Intel Pentium Pro "quad" 这样的基于总线的共享存储器多处理器上，用软件透明地支持共享地址空间通信的抽象。在一个 4 处理器节点中，以硬件的高效性支持一致的共享存储器；而跨节点时，一致的共享存储器则是用软件以低得多的效率支持的。给定一组应用和感兴趣的问题规模，你将要写一个评价你的系统的研究报告。你可能希望进行什么样的性能比较来理解这个跨节点体系结构的有效性？你会设计什么样的实验？每一种类型的实验将告诉你什么？你将使用什么样的指标？你有 16 个基于总线的多处理器节点，每个节点上有 4 个处理器，总共有 64 个处理器。假设你使用问题约束的扩放，而且你已经选择了问题的规模。

4.9　如本章所讨论的，在实际研究体系结构折中时通常使用两种类型的模拟：轨迹驱动的和执行驱动的。在轨迹驱动的模拟和执行驱动的模拟之间的主要折中是什么？在什么条件下，你认为结果（比如，一个程序的执行时间）会显著不同。

4.10　考虑多处理器模拟的难度和精度。

1）考虑处理器、存储系统、网络、通信辅助部件、时延、带宽或者竞争，你认为系统哪个方面最难被精确地模拟？哪个方面相对来说容易模拟？在每一种情况下，主要的困难是什么？在这诸多方面，你认为哪一些最重要，应该非常精确地加以模拟？而哪一些可以牺牲一些模拟的精度？

2）当试图评价通信体系结构的折中的影响时，考虑适当地模拟处理器流水线的重要性。尽管很多现代的处理器是超标量的和动态调度的，一个单条指令发射的静态调度的处理器的模拟却要容易得多。假设你想要模拟的实际处理器主频为 200 MHz，具有双指令发射能力，但实现了每条指令 1.5 个周期（1.5 CPI）的完美存储器。你能否在一个致力于理解网络传送时延的改变对端性能的影响的研究中，把它模拟成一个具有 300 MH 主频、指令单发的处理器吗？主要要考虑的问题是什么？

4.11　考虑你所熟悉的在一个二维网格上的最近邻居网格计算的迭代，使用子块分割。假设 264
我们使用四维数组的表示，其中前两维表示适当的分区，我们要评价的全规模问题是一个有双精度元素的 8 192 × 8 192 的网格，有 256 个处理器，每个处理器有 256 KB 的直接映射高速缓存，高速缓存块的尺寸为 128 字节。我们不能模拟这么大规模的问题，但是可以模拟一个 64 个处理器的 512 × 512 的网格问题。

1）你要选择的高速缓存的尺寸是什么？为什么这样选择？

2）列出选择太小的高速缓存会带来的危险。

3）你要选择什么样的高速缓存的块尺寸和相联性？会涉及到什么样的问题和禁忌？

4）在多大程度上你认为结果对于在更大机器上的全规模的问题有代表性？你会用这种配置去评价对某种通信体系结构的优化吗？可以评价该应用在机器上获得的加速比吗？

4.12　在像 Barnes-Hut 这样的星系模拟的科学应用中，影响扩放的关键问题是误差。这些应用经常模拟发生在自然界的物理现象，通过一些近似，用一个离散的模型来表示一个连续的现象并采用数值逼近的技术来解决问题。有几个应用参数代表了不同的近似的

源，因此也代表了模拟中的误差。比如，在 Barnes-Hut 中，粒子的数量 n 代表了对星系采样的精度（空间的离散化），时间步间隙 Δt 表示了时间离散化所做的近似，力计算精度参数 θ 决定了该计算的近似程度。运行较大问题的应用科学家的目标通常是在模拟中减少总体误差，并使它更加精确地反映被模拟的现象。尽管科学家调整不同的近似时没有通用的规则可循，但是对一个物理模拟来说，既具有直觉上的吸引力，又具广泛实践应用性的原则是：应该调整所有的误差源，这样它们的误差影响是相等的。

对于 Barnes-Hut 这样的星系模拟，星体物理学的研究（Hernquist 1987；Barnes 和 Hut 1989）表明，当某些误差影响不是完全独立时，下列规则在感兴趣的参数范围内是有效的：

- n：n 增加 s 倍会导致模拟误差降低$\sqrt{s}$倍。
- Δt：为了随时间而归并粒子轨道所使用的方法有一个 Δt^2 数量级的的全局误差。因此，误差降低$\sqrt{s}$倍（为了匹配 n 的 s 倍的增加）要求 Δt 降低$\sqrt[4]{s}$倍。这意味着要想让模拟持续保持不变的恒定量的物理时间，时间步数需要增加$\sqrt[4]{s}$倍。
- θ：在有实际意义的范围内，力的计算误差和 θ^2 成比例，因此，误差降低$\sqrt{s}$倍会要求 θ 降低$\sqrt{s}$倍。

265

假设在本习题中，某个规模的问题在 p 个处理器上的执行时间是在一个处理器上的执行时间的 $1/p$；也就是说，在问题约束扩放下，那个问题规模获得了完美的加速比。

1）如果 n 增加了 s 倍，你要如何调整其他的参数 θ 和 Δt？与只调整粒子的数量 n 的简单扩放对照，把这个规则称为*真实扩放*。

2）在串行程序中，数据集的尺寸和 n 成比例并和其他的参数无关。在共享地址空间中，在真实扩放和简单扩放下的存储器需求的增长是否不同（假定重要的工作集能容纳于高速缓存）？在消息传递型系统中会如何呢？

3）串行执行时间的增长大约是遵循

$$\frac{1}{\Delta t}\cdot\frac{1}{\theta^2}\cdot n\log n$$

（假设模拟固定量的物理时间）。如果 n 扩大 s 倍，假设完美加速比的情况，在真实扩放和简单扩放中，在 p 个处理器上的并行执行时间如何调整？

4）当处理器数量增长 k 倍，在 MC 扩放模型下，采用简单和真实两种扩放时，并行执行时间如何增长？如果在基本的机器上（未扩大规模之前）运行该问题需要一天，对于简单模型和真实模型下，在扩大的机器上运行该问题需要多长时间？

5）对于简单模型和真实模型，当处理器数目增加 k 倍，在共享地址空间中可以模拟的粒子数量在 TC 扩放下如何增长？

6）对于这个应用哪种扩放模型更加实际？MC 还是 TC？

4.13 对于 Barnes-Hut 这个例子，在真实和简单的 MC、TC 和 PC 扩放模型下，如何扩放下面的执行特征？

1）通信与计算的比率。首先假设它仅仅与 n 和处理器数量 p 有关，按照$\sqrt{p}/\sqrt{n}$变化，请在不同的模型下粗略的画出这个比值随处理器数量增长速率的曲线。然后说明

在不同的扩放模型下，其他参数可能产生的影响。

2）在共享地址空间的机器上为了获得好的性能所需要的不同工作集的规模和与此相适应的高速缓存的尺寸，请粗略地描绘在不同模型下，随着处理器数量的增加，最重要工作集的增长速率曲线。它在扩放方面补充了什么主要的方法上的结论？评价一下这些趋势和在消息传递型系统中局部基本树所需要的本地复制量的趋势有什么不同。

3）同步的频率（比如说每个单元计算一次），包括加锁和栅障两者。至少定量地加以描述。

266

4）输入/输出操作的平均频率和规模，假设每个处理器打印出所有分配给它的星体的位置 i）每十个时间步打印一次；ii）每隔固定量的模拟物理时间（比如，在星系演化中模拟时间中的每一年）。

5）在一致共享地址空间中的力的计算中，可能同时共享（访问）一个给定的星体数据的处理器的数量。（正如我们将在第 8 章中看到的，这个信息在为可扩展的共享地址空间的机器设计高速缓存一致性协议时有用处。）

6）在显式的消息传递型实现中消息的频率和尺寸。集中在力计算所需要的通信，假设每个处理器向其他每个处理器仅仅发送一条消息，该消息传送了后者在该时间步计算它的力需要从前者获得的数据。

4.14　基数分类应用需要并行的前序计算，从局部直方图计算出全局直方图。该计算的一个简化的版本如下。假设 p 个进程中的每一个进程有一个已经计算好的局部值（把这看作表示那个进程的直方图中对应于给定数位值的键字的数量）。我们的目标是计算一个有 p 个项的数组，项 i 是从进程 0 到进程 $i-1$ 所有局部值的总和。

1）描述并实现计算这个输出数组的最简单的线性方法。

2）现在设计一个具有较短关键路径的并行方法。（提示：你可以使用树结构。）分析每个方法所需要的时间。实现这两个方法，比较它们在你所选择的机器上的性能。在这里可以使用简化的例子或者更完整一些的例子，“本地值”实际上是一个数组，每个项是基数的一个数位，输出数组是二维的、由进程标识符和基数数位索引的数组。也就是说，更完整一些的例子对于每个基数数位都进行计算，而不是只计算一个基数数位。

267

3）讨论在后一个方法中你能够使用的协调同步的不同方法以及它们之间的折中。