# 第17章 并行处理

17.1多处理器组织

并行处理机系统的类型

平行组织

17.2对称多处理器

组织机构

多处理器操作系统设计考虑

17.3高速缓存一致性与MESI协议

软件解决方案

硬件解决方

MESI协议

17.4多线程和片上多处理器

隐式和显式多线程

显式多线程方法

17.5集群

集群配置

操作系统设计问题

集群计算机体系结构

刀片服务器

集群与SMP对比

17.6非均匀存储器存取

动机

组织机构

NUMA利弊

17.7云计算

云计算元素

云计算参考架构

17.8关键术语、思考题和习题

**学习目标**

学习本章后，你应该能够：

* 总结并行**处理器组织**的类型。
* 介绍对称**多处理器**的设计特点。
* 了解多处理器系统中的**cache一致性**问题。
* 解释**MESI协议**的关键特性。
* 解释**隐式**和**显式多线程**之间的区别。
* 总结**集群**的关键设计问题。
* 解释**非均匀内存**访问的概念。
* 概述**云计算**概念

传统上，计算机被看成是一个顺序机器。大多数计算机程序设计语言要求程序员指定作为指令顺序的算法。处理器执行程序是顺序地并且每次一条地执行机器指令。每条指令又是顺序地执行一系列操作（取指令、取操作数，完成运算，保存结果）。

这个计算机的顺序观点已不完全是真的了。在微操作级，多个控制信号是同时产生的。指令流水线中至少也要将取指和执行操作相重叠，这已不是新鲜事了。这两个例子都说明了独立操作的并行执行。这种方法进一步扩展到超标量组织，它发掘了指令级并行性，单个处理器内有多个执行单元，可并行执行同一程序的多条指令。

随着计算机技术的发展和硬件成本的下降，计算机设计人员开始寻求越来越多的实现并行的机会，通常用于提高性能，某些情况下用于改善可用性。在进行概述之后，本章将考察并行组织的几个最著名的方法。首先考察对称多处理器（SMP），这是最早采用，并且至今仍广泛使用的并行组织方法。在SMP组织中，多个处理器共享一个公共的存储器，由此带来了cache一致性问题，我们将用一节的篇幅专门讨论此问题。然后介绍集群系统，它是多个独立的计算机以一种协作风格组织而成。接着，本章考察多线程处理器和片上多处理器。对于工作负载超出单个SMP能力的情况采用集群系统方法变得日益普遍。使用多个处理器的另一种方法是非均匀存储器访问（NUMA）式机器，这是一种比较新的方法，有待市场的进一步验证，但它经常被看作是SMP或集群系统的替代方式。最后，本章着眼于云计算架构。

## 17.1多处理器组织

### 并行处理机系统的类型

最初由Flynn提出的一种分类方法［FLYN72］至今仍是最普遍使用的，是对具有并行处理能力的系统进行分类的方法。Flynn提出了下列计算机系统类型：

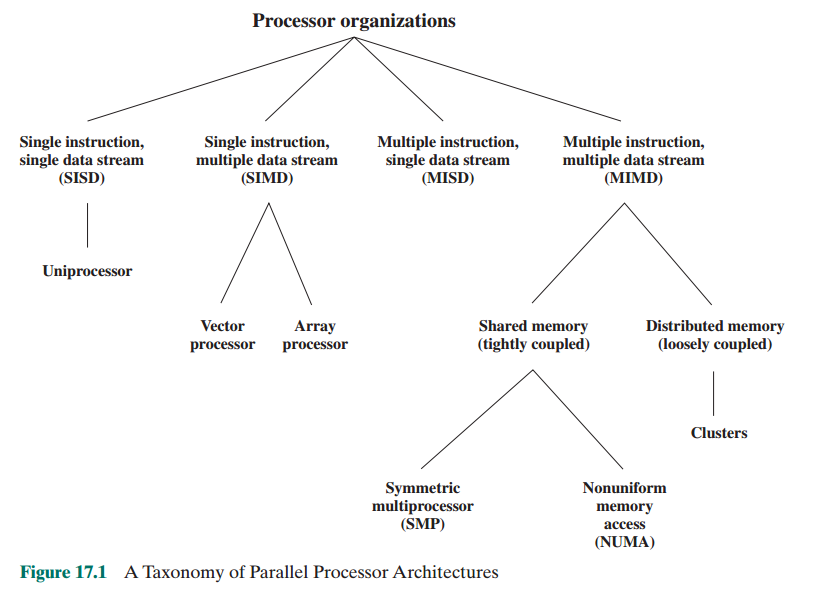
* **单指令单数据（SISD）流：**单一处理器执行单一指令流来操作保存于单一存储器上的数据，单处理器系统属于这一类。
* **单指令多数据（SIMD）流：** 一条机器指令控制几个处理部件基于锁步方式同时执行，每个处理部件有一个相关的数据存储器，故每条指令在不同的数据组上执行。17.7节讨论的向量和阵列处理器属于这一类。
* **多指令单数据（MISD）流：**一系列数据被发送到一组处理器，每个处理器执行不同的指令序列。这种结构从来没有在商业上被实现过。
* **多指令多数据（MIMD）流：**一组处理器同时执行不同的指令序列，对不同的数据集进行操作。SMP、集群系统和NUMA系统都属于这一类。

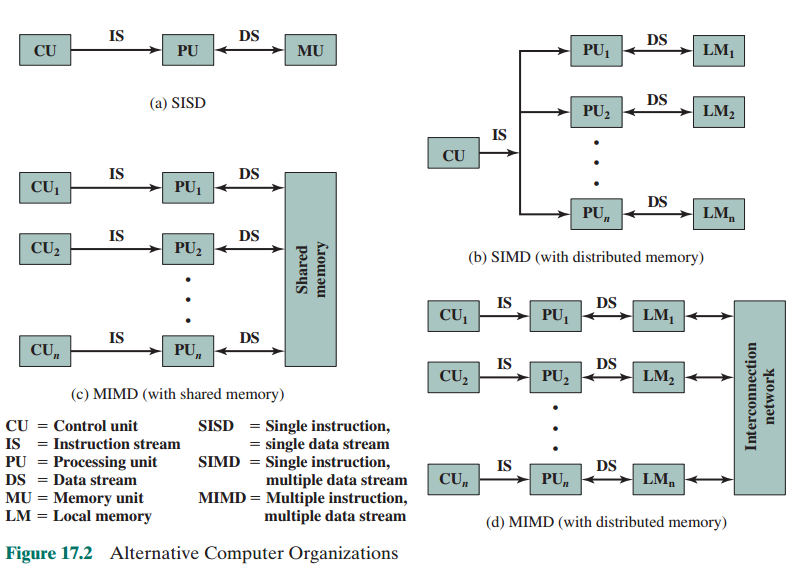
对于MIMD组织，处理器是通用的。每个处理器都能处理所有完成相应数据变换的必要指令。 MIMD能以处理器的通信方式来进一步划分（见图17-1）。如果处理器共享一个公共的存储器，则每个处理器存取共享存储器中的程序和数据，并经由此存储器相互通信。这种系统的最普通形式是**对称多处理器**（**symmetric multiprocessor，** **SMP**），我们将在17.2节考察它。在SMP中，通过共享总线或其他互连结构，多个处理器共享单一存储器或存储器池；一个明显特征是，存储器任何区域的存取时间，对各个处理器大致是相同的。当前的一种新的实现方法是**非均匀存储器访**（**non-uniform memory access，** **NUMA）**组织，我们将在17.5节介绍它。顾名思义，存储器中不同区域的存取时间，对一个NUMA的处理器是不同的。

一组独立的单处理器或SMP，可互连成**集群系统**（cluster），计算机之间的通信或是经由固定的路径，或是经由某些网络设施。

### 并行组织

图17-2说明了图17-1分类的通常组织。图17-2a表示一个SISD的结构。这里有某种控制单元（Control Unit， CU）向处理单元（Processing Unit，PU）提供一个指令流（Instruction Stream，IS），该处理单元对来自一存储单元（Memory Unit，MU）的单一数据流（Data Stream， DS）进行操作。对SIMD而言，还是一个单一控制单元，不过现在是向多个处理部件提供单一指令流，每个处理部件可有自己的专用存储器（见图17-2b）或者有一个共享的存储器。最后是MIMD，它有多个控制单元。每个向自己的处理部件提供一个独立的指令流。MIMD可以是共享存储器的多处理器（见图17-2C），或是一个分布式存储器的多计算机（multicomputer）（见图17-2d）。





SMP、集群系统和NUMA机器的设计是一个涉及物理组织、互连结构、处理器问通信，操作系统设计和应用软件技术的复杂问题。这里，我们关注的主要是组织。不过也将简要介绍操作系统设计问题。

## 17.2对称多处理器

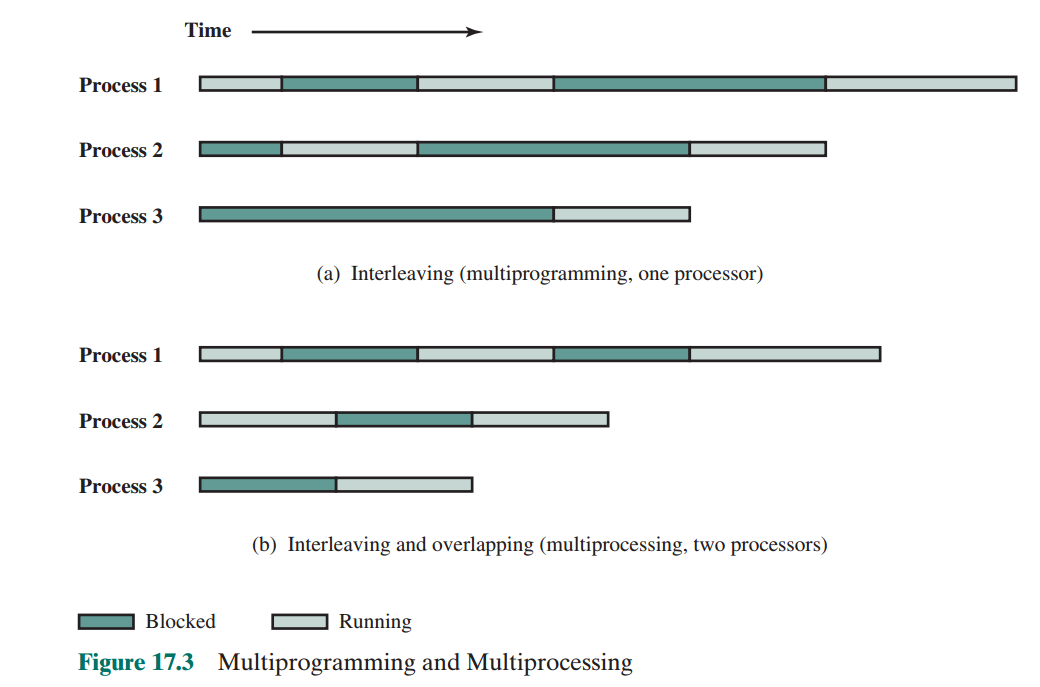
直到前不久，几乎所有的单用户个人计算机和大多数工作站还只含有单一通用微处理器，随着对性能需求的增长和微处理器价格的持续下跌，促使厂商推出了SMP系统。SMP既指计算机硬件体系结构，也指反映此体系结构的操作系统行为，可把SMP定义为具有如下特征的独立计算机系统：

1. 有两个或更多功能相似的处理器。
2. 这些处理器共享同一主存和I/O设备，以总线或其他内部连接机制互连在一起；这样，存储器的存取时间对每个处理器大致都是相同的。
3. 所有处理器共享对I/O设备的访问，或通过同一通道，或通过提供到同一设备路径的不同通道。
4. 所有处理器能完成同样的功能（术语"对称"的由来）。
5. 系统被一个集中式操作系统（OS）控制，OS提供各处理器及其程序之间的作业级、任务级、文件级和数据，元素级的交互。

第（1）点到第（4）点是不言而喻的，第（5）点表示了与集群系统之类的松耦合多处理系统的对照。后者所交互操作的物理单位通常是消息或整个文件；而SMP中，各个的数据元素能成为一个交互级别，于是处理器间能够有高度的相互协作。

SMP的操作系统能跨越所有处理器来调度进程或线程。SMP有如下几个超过单处理器的潜在优点：

* **性能（performance）：**如果可以对一台计算机完成的工作进行组织，使得某些工作部分能够并行完成，则具有多个处理器的系统与具有同样类型的单处理器的系统相比，将产生更高的性能（见图17-3）。



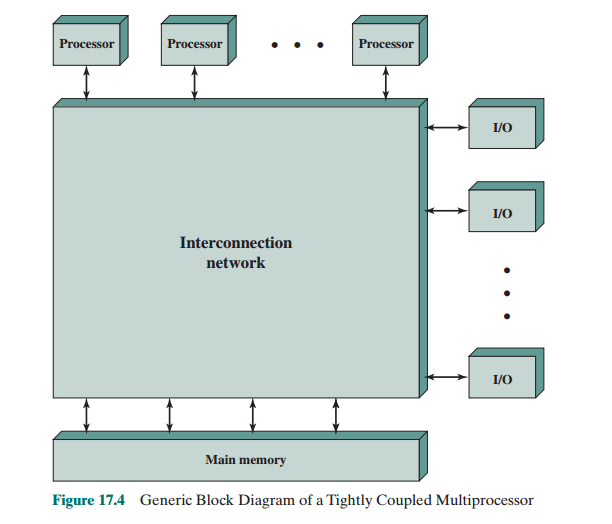
* **可用性（availability）：**在一个对称多处理器中，所有处理器都能完成同样的功能。故单个处理器的故障不会造成系统的停机，系统可在性能降低的情况下继续运行。
* **增量式增长（incremental growth）：**用户可以通过在系统中添加处理器来提高系统性能。
* **可扩展（scaling）：**厂商能提供一个产品范围，它们基于系统中配置的处理器数B不同而  
  有不同的价格和性能特征。

应当注意，这些只是潜在的优点，而不是必有的，操作系统必须提供相应工具和功能，以开发SMP系统中的并行性。

SMP的一个有吸引力的特点是，多个处理器的存在对用户是透明的；由操作系统实际管理各个处理器上的进程或线程的调度，以及处理器间的同步。

### 组织

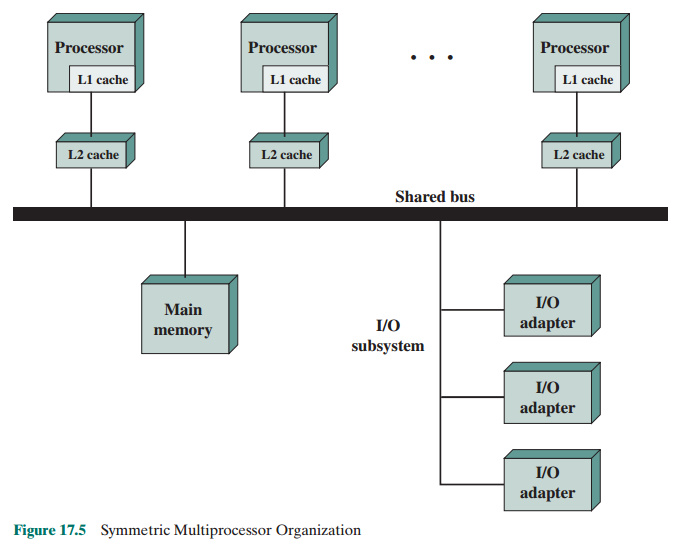
图17-4说明了多处理器系统的一般组成情况，其中可以有两个或更多的处理器。每个处理器是自包含的，即包括有控制器、ALU、寄存器等。通常，还会有一级或多级cache。通过某种形式的互连结构，每个处理器能访问共享的主存储器和1/0设备。各处理器之间通过存储器（位于公共数据区中的消息或状态信息）能相互通信。处理器之间直接交换信号是可能的。存储器经常组织成允许对存储器各个块的多重同时存取。在某些配置中，每个处理器除了共享资源之外，还允许有它专用的主存储器和I/O通道。



对个人计算机、工作站和服务器而言，最普通的组织是分时共享总线。分时共享总线是构成一个多处理器系统的最简单结构（见图17-5）。分时共享总线系统的结构和接口基本上同于使用总线互连的单处理器系统。总线由控制、地址和数据线组成。为便于I/O子系统到处理器的DMA传送，应提供如下特征：

* **寻址**：必须能区别总线上各模块，以确定数据的源和目标。
* **仲裁：**任何**I/O**模块都能临时行使主控器功能。因此需要提供一种机制来对总线控制的竞争请 求进行仲裁，这可使用某种类型的优先级策略。
* **分时复用**：当一个模块正在控制总线时，其他模块是被锁住的，而且需要的话，应能挂起它的操作直到当前的总线访问被完成。

这些单处理器特征在对称多处理器（SMP）配置中是直接可用的。稍后将会看到，SMP中有多个处理器以及多个I/O处理器都试图掌管总线，对一个或多个存储器模块进行访问的更为复杂的情况。

与其他方法比较，总线组织方式有如下几个优点：

* **简易性：**这是多处理器系统组成的最简单方式。物理接口以及每个处理器的寻址、仲裁和分时逻辑可与单处理器系统保持相
* **灵活性：**可以通过附加更多处理器到总线的方式来扩充系统，这一般来说也是容易的。
* **可靠性：**本质上来说，总线是一个被动介质，并且总线上任一设备的故障不会引起整个  
  系统的失败。

总线组织的主要缺点在于性能。所有的存储器访问都要通过公共总线，于是系统速度受限于总线周期时间。为改善性能，就要求为每个处理器配cache，这将有效地减少总线访问次数。一般来说，工作站和PC的SMP组织都有两级cache。L1 cache是内部的（与处理器同一芯片），L2 cache或是内部的或是外部的。现在，某些处理器还使用了 L3 cache。

cache的使用导致某些新的设计考虑，因为每个局部cache只保存部分存储器的映像，如果在某个cache中修改了一个字，可想象出其他cache中的此字将会是无效的。为防止这个问题，必须通知其他处理器：已经发生了修改。这个问题称为cache一致性（cache coherence）问题，并且一般是用硬件解决，而不是由操作系统去解决，17.4节将专门讨论这个问题。

### 多处理器操作系统设计考虑

一个SMP操作系统（OS）管理多个处理器和其他计算机资源，而使用户感觉到单一操作系统控制着系统资源。事实上，这种配置应呈现为一个单处理器多道程序设计系统。不论是SMP还是单处理器情况，都可以一次启动多个作业或进程；调度它们的执行和分配资源是操作系统的责任。用户可构造使用多进程或多线程的应用程序，而无需顾及程序是在单一处理器还是在多个处理器上进行。于是，多处理器操作系统必须提供多道程序设计系统的所有功能。以及伴随多个处理器而来的其他特征。关键的设计问题如下所示：

* **同时并发进程（simultaneous concurrent process）：**OS例程必须是可重入的，以允许几个处理器可同时执行同一 OS代码。当多个处理器执行OS的同一部分或不同部分时。要恰当地管理OS的表和其他数据结构，以避免死锁或无效操作。
* **调度（scheduling）：**任何处理器都可完成调度，因此必须避免冲突。调度程序必须将就绪进程指派给可用处理器。
* **同步（synchronization）：**由于有多个活动进程可能访问共享地址空间，或共享Ｉ/O资源，因此必须提供有效的同歩。同步是一种机制，它保证相互排斥的访问和事件的次序。
* **存储管理（memory management）：**正如第8章所讨论的，多处理器上的存储管理必须解决单机系统上出现的所有问题。此外，操作系统还要发掘可用的硬件并行性，如多端口存储器，以实现最佳性能。操作系统必须协调不同处理器上的分页机制，以保证几个处理器共享页（段）时的一致性，并在发生页替换时决定合适的操作。
* **可靠性和容错（reliability and fault tolerant）：**面对处理器故障，操作系统应提供优雅的降级使用。调度程序和操作系统的其他部分必须识别处处理器的失效，并相应地重构管理表。

## 17.3 cache 一致性和MESI协议

当代多处理器系统中，通常是每个处理器都有一级或两级cache。这种组织是出于性能的考虑，然而也产生了一个称为cache一致性的问题。此问题是：同一数据的多个副本可能同时存在于不同的cache中。若允许处理器自由地修改它们自己的副本，就会导致不同cache对存储器中同一数据的反映不一致。第4章曾定义了两种通用的写策。

* **回写（write back）：**写操作通常只是对cache进行，仅当此cache数据行由cache换出时，相应的主存内容才被修改。
* **写直达（write through）：**所有写操作既对cache也对主存进行，保证主存总是有效的。

很显然，回写法会导致不一致。如果两cache保存同一行数据，一个cache修改了此行，另一个cache将不知道它保存的已是过时数据，尔后读此行数据，将产生一个无效结果。即使采用写直达法，也会出现不一致，除非其他cache监督存储器的访问情况，或接收某些直接的修改指示。

本节我们先简要介绍解决cache 一致性问题的几种方法，然后集中介绍最广泛使用的方法：MESI协议。这个协议的版本已被用于x86体系结构中。

对任何cache一致性协议，目标都是让近期使用的局部变量进人cache，并保留在那里，经历多次读和写，而使用协议来维护同时可出现在多个cache中的共享变量的一致性。ache一致性方法通常可分为软件方法和硬件方法。某些实现采用了二者相结合的策略。无论如何，分类成软件方法和硬件方法是有指导意义的，并广泛用于评价cache 一致性。

### 软件解决方案

cache一致性问题的软件解决方法的思路是：依赖于编译程序和操作系统来解决问题，力图避免附加硬件电路和逻辑。因为确定潜在不一致问题的开销由运行时转换成编译时，设计的复杂性由硬件转移到软件，故软件解译方法是很有吸引力的。从另一方面看，编译时软件方法一般采取的是保守性的判决，这会导致cache利用率的下降。

基于编译程序的一致性机制通过进行代码分析，来确定什么样的数据项对于高速缓存可能会变成不安全的，然后相应地标记出这些项。操作系统或硬件来防止这些不可高速缓存的项进入cache。

最简单的方法是不准任何共享数据变量被高速缓存。但这种方法太保守了，因为一个共享数据结构可以在某些时期内是排他性使用的，并可在某些其他时期内是只读的。只有在如下的期间内：即至少一个进程去修改变量而同时至少有另一个进程会去访问此变量，才会出现cache一致性问题。

更有效的方法是分析代码来确定共享变量的安全期间。然后，编译程序在生成代码中插入指令，以在临界期间实施cache一致性的雏护。已研制了几种技术用于代码分析和保证结果的正确性，详见［LILJ931］和［STEN90］。

### 硬件解决方案

基于硬件的解决方法通常称为cache一致性协议，它提供了对运行时潜在的不一致情况的动态识别。因为是在问题实际发生时及时解决，所以cache的使用更有效，性能的改善要比软件解决方法好。另外，这些方法对程序员和编译程序都是透明的，也减轻了软件开发负担。

不同的硬件解决方法在几个具体细节上有所不同，包括数据行状态信息保存在何处，信息是如何组织的，在何处实施一致性维护以及实施结构又是怎样的等。通常，硬件解决方法分成两大类：**目录协议（directory protocol）**和**监听协议（snoopy protocol）**。

1. **目录协议**

目录协议收集并维护有关数据块副本驻存在何处的信息。典型地，系统有一集中式控制器，它是主存控制器的一部分，目录就存于主存中。目录含有关于各个局部cache内容的全局性状态信息。当某个cache控制器产生一个访问请求时，集中式控制器检查此请求并发出必要的命令，以在存储器和cache之间，或cache相互之间传送数据。它亦负责保持状态信息的更新。于是，任何一个能影响cache数据行全局状态的局部动作必须报告给中央控制器。

一般来说，控制器维护着关于哪个处理器拥有哪个数据行副本的信息。在处理器向局部的cache行副本写入之前，它必须向控制器请求排他性存取权。在同意这次排他性存取之前，控制器发送一个消息给所有包含该行副本高速缓存的处理器，以强迫每个处理器使它的副本无效。接收到这些处理器返回的认可后，控制器才将排他性存取权授给请求的处理器。当一行已授权给某处理器专有，而另外的处理器企图读此行时，它将送出一个未命中提示给控制器。控制器则向拥有此行的处理器发出命令，要求它此行写回到主存。于是，现在此行可被原先的处理器和请求的处理器读共享了。

目录协议有中央瓶颈的缺点，另外各cache控制器和中央控制器之间的通信开销也大。然而，在采用了多条总线或某种另外的复杂互连结构的大型系统中，它们是很有效的。

1. **监听协议**

监听协议将维护cache 一致性的责任分布到多处理器中各个cache控制器上。一个cache必须识别出何时它保有的一行是与其他cache共享的。当对共享的cache行完成一个修改动作时，它必须通过一种广播机制通知所有其他cache。各个cache控制器应能监听互联网络，以得到这些广播通知，并做出相应的反应。

监听协议非常适合于基于总线的多处理器，因为共享的总线能为广播和监听提供简洁的方式。然而，使用局部cache的一个目标就是希望避免或减少总线访问，因此必须小心地设计，不使由于广播和监听而增加的总线传输开销抵消掉使用局部cache的获益。

监听协议已开发出两种基本方法：写-作废（write-invalidate）和写-更新（write-update）方法。使用写-作废协议，系统任一时刻可有多个读取者。但只有一个写入者。最初，一个数据行可能在几个cache中为读目的而共享。当某个cache要对此行完成一个写操作时，它要先发出一个通知，以使其他cache中此行变为无效，使此行成为要执行写操作的cache专有。一旦cache行变为专有，拥有者处理器就可进行本地写操作，直到某些其他处理器要求此同一数据行。

写-更新协议又称为写-广播（write-broadcast）协议。根据这样的协议，系统可有多个写入者以及多个读取者。当一个处理器打算修改一个共享行时，欲被修改的特定数据字亦被广播到所有其他cache，于是包含此行的cache能同时进行写入修改。

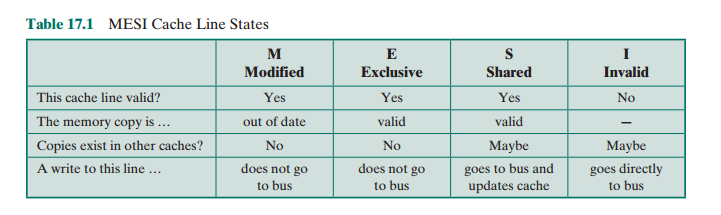
两种方法谈不上哪个更好些，性能取决于局部cache的数量和存储器读、写的样式。某些系统的实现既可适用写-作废协议，也可适用写-更新协议。

写-作废协议已广泛用于像x86体系结构这样的商业多处理器系统中。它（使用cache标记中的额外两位）标出每个cache行的状态是修改（Modified）、专有（Exclusive）、共享（Shared）还是无效（Invalid）。因此，写-作废协议也被称为MES1协议。本节的剩余部分。我们将考察它在多处理器的各局部cache中的使。出于简化的考虑，我们将不考察涉及局部的以及经过分布式多处理器的L1和L2 cache协调机制。如果对这些协调机制进行考察，也不会使我们了解更多的原理，但会使讨论变得很复杂。

### MESI 协议

为维护SMP中的cache 一致性，数据cache通常支持MESI的协议。根据MES1协议，数据cache的每行包括一个两个状态位的标记（tag），这样每行可处于下列4种状态之一：

* **修改态（modified）：**此cache行已被修改（不同于主存），并仅在这个cache。
* **专有态（exclusive）：**此cache行同于主存，但不出现于任何其他cache中。
* **共享态（shared）：**此cache行同于主存，并可出现于另外的cache中。
* **无效态（invalid）：**此cache行不含有效数据。



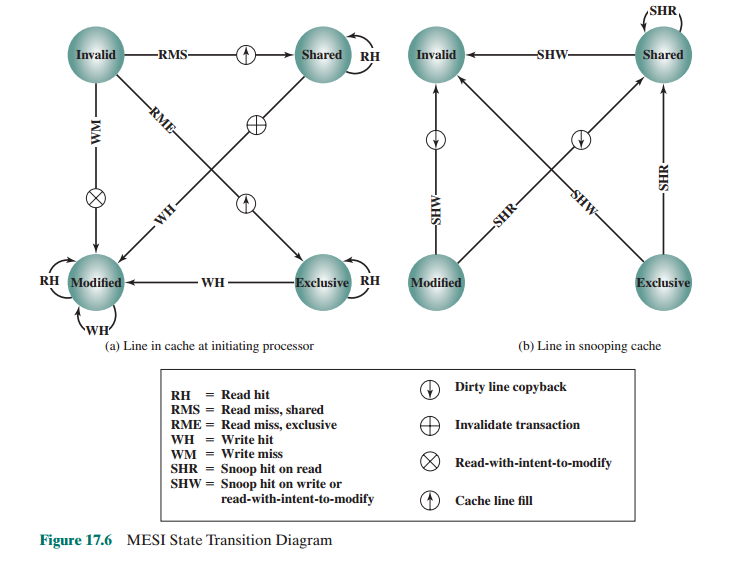


表17-1总结了 4种状态的意义。图17-6展示了 MESI协议的状态图。注意，每个cache行有自己的状态位，因此每个cache行本身的状态变化就实现了所示的状态图。图17-6a表示的是，由于连接这个cache的处理器发起动作而发生的状态转换。图I7-6b表示的是，由于监听到公共总线上的事件而发生的状态转换。对处理器发起的和总线发起的动作，分别给出状态图，这样有助于弄清楚MESI协议的逻辑。注意，任何时候cache行只处于单一状态，如果下一个事件来自所连接的处理器，那么状态转换由图17-6a给出；如果下一个事件来自总线，那么状态转换由图17-6b给出。下面让我们更详细地考察这些转换。

1. **读缺失**

当一个局部cache出现读缺失时，处理器启动一个存储器读，以读取包含有此未命中地址的主存行。处理器在总线上发出一个信号，通知所有其他处理器／cache单元监听此事务。这会有几种可能出现：

* 若另一处理器有此行的未修改副本（由主存读入后未被修改过），并处于专有态，则它返回一个信号，指示它共享此行。发出响应的处理器然后将它的副本状态由专有转换成共享。启动读操作的处理器由主存读入此行，并将此行的状态由无效转换成共享。
* 若一个或多个cache有处于共享状态的此行的干净副本，则它们中每个都指示共享此行。启动读操作的处理器由主存读入此行，并将此行的状态由无效转换成共享。
* 若另一 cache有此行修改过的副本，则此cache阻塞存储器读，并将此行经由总线提供给发出请求的cache。发出请求的cache 则把此行的状态从无效改变为共享１。发送给请求cache的数据行也会被存储控制器接收并处理，将其保存到存储器中。
* 若任何其他cache都无此行的副本（新的或修改过的），则没有信号返回。启动处理器读入此行，并将此行状态由无效转换成专有。

1. **读命中**

当读命中出现在当前局部cache行时，处理器简单地读取所需数据项，没有任何状态改变；仍保留在修改、共享或专有状态。

1. **写缺失**

局部cache出现写缺失时，处理器启动一个存储器读，来读取含有此末命中地址的主存行。为了写入修改的目的，处理器在总线上发出一个意为“用于修改的读”（read： with-intent-to-mod-ify，RWITM）信号，当此行装入后，它立即标记为修改态，并进行写入修改。依据其他cache状况，在进行数据行装入时会有两种情况。

第一种情况，其他某个cache有此行的修改副本（状态为修改态）。这种情况下，被通知的处理器告诉启动处理器，它有此行的修改过的副本；启动处理器放弃总线并等待；而被通知的处理器获得总线访问权，并将它修改过的副本写回主存，并将此cache行状态转换成无效（因为启动处理器正要修改此行）。接着，启动处理器再一次在总线上发出RWITM信号，然后从主存中读取该行，修改cache中的行，并把该行标识为修改状态。

第二种情况，任何其他cache都没有所要求行修改过的副本，也就没有信号返回，启动处理器着手它请求的行读入并修改的操作。在此期间，若一个或多个处理器有此行的干净副本并处于共享状态，则每个处理器都要将它的副本变为无效；若一个处理器有此行的干净副本而处于专有状态，则它同样要将此副本作废。

注释１――――――

　　在一些实现中，拥有修改过数据行的cache会向启动处理器发出一个重试信号。此时，拥有修改过数据副本的处理器获取总线控制权，将修改过的数据行副本写回到主存中，并将其cache中该数据行状态从修改转换到共享。接下来，原先发出请求的处理器再次发出读请求，这次会发现一个或多个处理器拥有该数据行处于共享状态的干净副本，于是按照前一点描述的情况操作。

1. **写命中**

当写命中出现在当前局部cache行上时，效应取决于局部cache中此行的当前状态：

* **共享：**完成写修改之前，处理器必须先得到此行的专有权。处理器在总线上通告它的打算，cache中含有此行共享副本的各处理器将各自的cache行状态由共享变成无效。然后，启动处理器完成写修改，并将它的cache行副本状态由共享转换成修改。
* **专有：**处理器已有对此行的排他性控制，故它只需要简单地完成写修改，并将此行的状态由专有变为修改。
* **修改：**处理器已有对此行的排他性控制，并已标志此行为修改态。它只需要简单地完成写修改。

1. **L1-L2 cache 的一致性**

至此，本书所介绍的cache 一致性是通过连接到同一总线或其他SMP互连结构上的各cache之间的协同动作来实现的。一般而言，这些cache是L2 cache，而每个处理器还有一个L1 cache。它不直接连到总线上，因而也不能直接参与监听协议的活动。于是，需要某种策略来维护SMP配置中跨越两级cache和跨越所有cache的数据完整性。

策略是扩展MESI协议（或任何其他cache一致性协议）到L1 cache，于是.每个L1 cache行都包括指示状态的位。本质上，目标如下：对于既出现在L2 cache中又出现在相应L1 cache中的仟何行，L1行的状态应追踪L2行的状态。实现此目标的最简单方式是在L1 cache使用写直达策略；这种情况下，写直达是到L2 cache而不是到存储器。L1写直达策略迫使对L1行的任何修改都送出到L2 cache；于是，使这一修改对其他L2 cache都是可见的。L1使用写直达策略要求L1的内容必须是L2内容的子集。这又暗示L2 cache的关联度应等于或大于L1 cache的关联度。IBM S/390 SMP采用了 L1写直达策略。

如果L1 cache使用了回写策略，两个cache间的关系将会更复杂。已有几种方法用于维护这种情况下的cache一致性，这个话题超出了我们的范围。

## 17.4多线程和片上多处理器

处理器性能一个重要的评测指标是它执行指令的速率。这可表示成：



这里，*f*是处理器时钟频率，单位为MHz；（instructions per cycle）是平均每周期执行指令数。于是，设计者在两个方面追求增强性能的目标：提高时钟频率和提高平均每周期执行的指令数，或严格地说，即提高在单个处理器周期期间所完成的指令数。正如在前面几章所看到的，设计者通过使用指令流水线和在超标量体系结构中使用多条并行的指令流水线办法来提高IPC。在流水线或多条流水线设计中，其原理性问题是各个流水段利用的最大化。为提高吞吐率，设计人员提出了更为复杂的机制，例如以不同于指令流顺序的次序执行某些指令，以及猜测执行一些可能不会被执行的指令。但正如2.2节所讨论过的那样，由于复杂性和所涉及的功率消耗，这种办法最终会达到一个限制点。

一种替代的办法是多线程化（multithreading），它允许高度的指令级并行而不增加电路复杂性和功率消耗。从本质上讲，这种办法是将指令流分成几个更细小的流，称为线程，这样，这些线程可能并行执行。

在商业系统和实验系统中所实现的各种专用多线程设计类型非常丰富。本节简要介绍其主要概念。

### 隐式和显式多线程

多线程处理器讨论中所使用的线程概念，可能同于也可能不同于多道程序操作系统设计中的软件线程概念。简要定义如下术语将是有用的：

* **进程（process）：**在计算机上运行的一个程序实例，它体现出如下两个主要特征。
  + **资源占有 （resource ownership）：**进程包括一个包含进程映像的虚拟地址空间；进程映像是定义进程的程序、数据、桟和属性的集合。随着时间的推移，进程会分配到如主存、I/O设备和文件这样资源的控制或占有权。
  + **调度/执行（scheduling/execution）：**一个进程的执行沿一条执行路径（踪迹） 流经一个或多程序。这个执行可能与其他进程的执行相交错。于是，一个进程具有各种执行状态（运行、就绪等）和派发（dispatch）优先权。进程是操作系统调度和派发的一个实体。
* **进程切换（process switch）：**将处理器由一个进程转换到另一个进程的操作。它通过保存第一个进程的所有控制数据、寄存器和其他信息，并以第二个进程的信息来替换它们而完成２。3
* **线程（thread）：**进程内可派发的任务单位。它包括一个处理器上下文（程序计数器和栈指针等）和它专有的栈数据区域（以允许子例程转移）。线程顺序执行并可中断，这样处理器能转向另一线程。
* **线程切换（thread switch）：**在同一进程中，将处理器控制由一线程转换到另一线程的动作。

通常，这类切换的代价要比进程切换少得多。于是，线程关注的只是调度/执行，而进程关注的是调度/执行和资源古有。一个进程中的多个线程共享同样资源。这是为什么线程切换的耗时要比进程切换少得多的原因。像UNIX早期版本这样的传统操作系统不支持线程。大多数当代操作系统，如Linux、UNIX后期版本、Windows等，都支持线程。应区分用户级线程和内核级线程，前者对应用程序是可见的，后者仅对操作系统是可见的。两者都称为显式线程，因它们是软件中定义的。

所有商业处理器和大多数实验处理器至今使用的都是显式多线程。这些系统并发执行来自不同显式线程的指令，或通过在共享流水线上交错执行来自不同线程的指令，或通过在并行流水线上的并行执行来实现多线程。隐式多线程化指的是，从单个顺序程序中提取多个线程来并发执行。这些隐式线程可静态地被编译器定义或动态地被硬件定义。本节的其余部分只考虑显式多线程化。

### 显式多线程的方式

首先，多线程式处理器必须为并发执行的每个线程提供一个分立的程序计数器。不同的设计区别在于为支持并发多线程执行所添加的硬件类型和总量。通常，以线程为基础来进行取指令操作。处理器分别对待每个线程，并可用几种技术来优化单线程的执行，这包括分支预测、寄存器重命名和超标量技术。我们要实现的是线程级并行性，它与指令级并行性结合后会提供极大的性能改善。

概括来说，多线程化有如下4种办法：

* **交错式多线程**（interleaved multithreading）：这亦称为**细粒度多线程** （fine-grained multithreading） 。处理器同时处理两个或多个线程上下文，每个时钟周期由一个线程切换到另一个线程。若由于数据相关性或存储器等待而使一个线程阻塞，则跳过此线程去执行另一个就绪线程。
* **阻塞式多线程**（blocked multithreading）：这亦称为**粗粒度多线程**（coarse-grained multithreading）。线程的指令连续地执行，直到如cache缺失这类引起延迟的事件出现。这类事件引发处理器切换到另一线程。这种办法在按序执行的处理器上是比较有效的，因为像cache缺失这类的延迟事件会使流水停顿。

注释2---------------------------------

术语”上下文切换”（context switch）通常出现在有关搡作系统的文献和教材中。不幸的是，虽然大多数文献使用该术语表示这里所说的进程切换，但是另外一些文献使用该术语来表示线程切换，因此，为避免歧义，此处使用进程切换而非上下义切换。

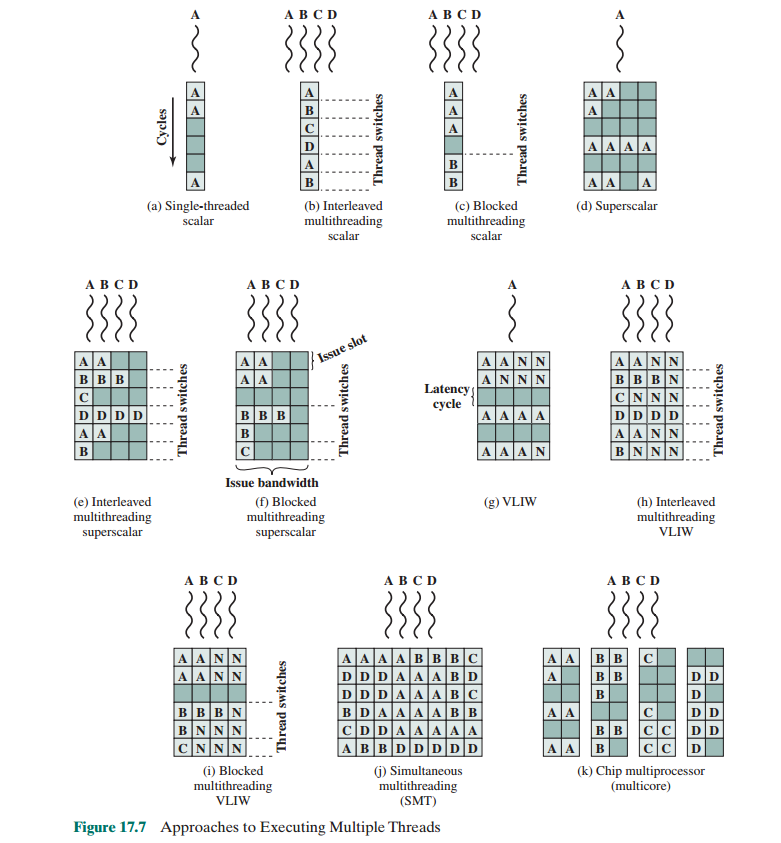
* **同时多线程**（simultaneous multithreading，SMT）：来自多个线程的指令同时发射到超标量

　　处理器的执行单元。它将超标量多条指令发射的能力与多个线程上下文的使用相结合。

* **片上多处理**（chip multiprocessing）：这种情况是完整处理器在单一芯片上进行复制，每个处理器处理一个分立线程。这种办法的优点是，能有效地使用芯片上可用逻辑面积而不依赖复杂性日益增长的流水线设计。

对于前两种办法，来自不同线程的指令不是同时被执行的，而是通过使用不同的寄存器组和其他上下文信息，处理器能快速地由一个线程切换到另一个线程。这使得处理器的执行资源更好地得到利用，并避免了由于cache缺失和其他延迟事件所造成的较大性能损失。SMT办法利用被复制的执行资源，完成了来自不同线程的指令真正地同时被执行。片上多处理亦允许来自不同线程的指令同时执行。

基于［UNGE02］的图17.7展示了一些涉及多线程化的可能的流水体系结构，并将它们与不使用多线程的办法相对照。每个水平行代表一个执行周期可有的发射槽３；即每行的宽度对应于单一时钟周期所能发射的最大指令数。垂直方向表示时钟周期的时间顺序。一个空槽（画有阴影）代表在流水线中未被用于执行的槽。用N表示无操作（no-op）。



注释３-------

发射槽（issue slot）是在一给定时钟周期内可发射指令所来自的位置。回顾第14章中指令发射的介绍，指令发射是在处理器功能单元上启动指令执行的过程。这发生在指今从流水线的译码阶段向流水线执行阶段第一级前进的时候。

图17-7的前三个说明了标量（即单发射）处理器所使用的不同办法：

* **单线程标量**（single-threaded scalar）：这是在传统RISC和CISC机器上见到的简单流线，没有多线程。
* **交错式多线程标量（**interleavedmultithreadedscalar**）**：这是最易实现的多线程化办法。通过每时钟周期切换线程，使流水线各段保持或接近满载。硬件必须在周期之间能由一个线程上下文切换到另一线程上下文。
* **阻塞式多线程标量**（blocked multithreaded scalar）：这种情况下，单线程连续执行，直到使流水停顿的等待事件出现，此时处理器切换到另一线程。

图17-7c表示的完成线程切换时间是一个周期的情况，而图17-7b表示线程切换以零周期实现。在交错式多线程情况中，假定了线程间无数据相关性和控制相关性，这就简化了流水线设计，并允许线程切换无任何延迟。然而，这取决于特定的设计和实现，阻塞式多线程可能会要求以一个时钟周期来完成线程切换，正如图17-7所示。若取来的指令触发线程切换，且必须由流水线逐出，就发生这种情况［UNGE03 ］ 。

虽然交错式多线程办法比阻塞式多线程呈现出更好的处理器利用率，但它这样做是以牺牲单线程性能为代价。多个线程竞争cache资源，这会使其中某些线程缺失cache概率升高。

若处理器每周期能发射多条指令，则有更多并行执行机会可用。图17-7d至图17-7i说明了几种类型的处理器，它们都具有每周期发射4条指令的硬件。在所有这些情况中，一个周期仅能同时发射来自单个线程的多条指令。下面分别予以说明：

* **超标量**（superscalar）：这是基本的超标量，无多线程。直到前不久，这还是在处理器内提供并行性的最强有力办法。注意在某些周期期间，不是全部的可用发射糟都被使用。在一些周期，少于最大数的指令被发射，这称为水平损失（horizontal loss）。在另一些周期，无发射槽可用；这些是有指令却不能被发射的周期，这称为垂直损失（vertical loss ）。
* **交错式多线程超标量**（interleaved multithreading superscalar）：每个周期期间，来自单一线程的尽可能多的指令被发射。正如前面所讨论过的，使用这种技术可消除由线程切换所带来的潜在延迟。然而，任一给定周期发射的指令数仍受限于给定线程内所具有的指令间相关性。
* **阻塞式多线程超标量**（blocked multithreaded superscalar）：再一次，任何周期仅来自单个线程的指令会被发射，并使用了阻塞式多线程。
* **超长指令字**（very long instruction word ， VLIW）： 一个 VLIW 体系结构，如IA-64，是将多条指令放在单个字中。典型地，由编译器构造VLIW，它把可并行执行的操作放在同一字中。在一个简单的VLIW机器中（图17-7g），如果不能以可并行发射的指令填满字，就使用无操作（no-op）填充。
* **交错式多线程超长指令字**（interleaved multithreading VLIW）：这个办法所提供的效能类似于交错式多线程在超标量体系结构上所提供的效能。
* **阻塞式多线程超长指令字**（blocked multithreaded VLIW）：这个办法所提供的效能类似于阻塞式多线程在超标量体系结构上所提供的效能。

图 17-7中所历的最后两种办法允许多个线程同时并行执行：

* **同时多线程**（simultaneous multithreading）：图17-7j表示一个每次能发射8条指令的系统。若一个线程具有高度的指令级并行性，它可在某些周期填满水平槽。在另一些周期，来自两个或多个线程的指令可被发射。若有充足的活跃线程，每周期发射最大数目的指令通常是可能的，这提供了更高级的效能。
* **片上多处理器**（chip multiprocessor）或**多核**（multicore）：图17-7k表示一个芯片含有4个处理器，其中每个都是一个双发射的超标量处理器。每个处理器各指派给一个线程，它能每周期发射来自此线程的两条指令。

比较图17-7j和图 17-7k，我们看到一个与SMT有同样指令发射能力的片上多处理器，却不能达到与SMT同等程度的指令级并行性。这是因为片上多处理器不能隐藏由发射来自其他线程的指令所导致的延迟。另一方面，片上多处理器应胜出有同样指令发射能力的超标量处理器，因为超标量处理器的水平损失将更大。此外，片上多处理器上的每个处理器使用多线程是可行的，当今某些机器正是这样做的。

## 17.5集群

计算机系统设计的一个重要而且较新的发展是集群化（clustering）。集群化作为一种提供高性能和高可用性的方法，是对称多处理器（SMP）的替代物，并对服务器应用特别有吸引力。我们将集群系统（cluster）定义成一组完整的计算机相互连接，它们作为一个统一的计算资源一起工作，并能产生好像是一台机器的假象。术语“完整计算机”（*whole computer*）意指一台计算机离开集群系统仍能执行自己的任务。在文献中，集群系统中的每台计算机一般称为结点（*node*）。

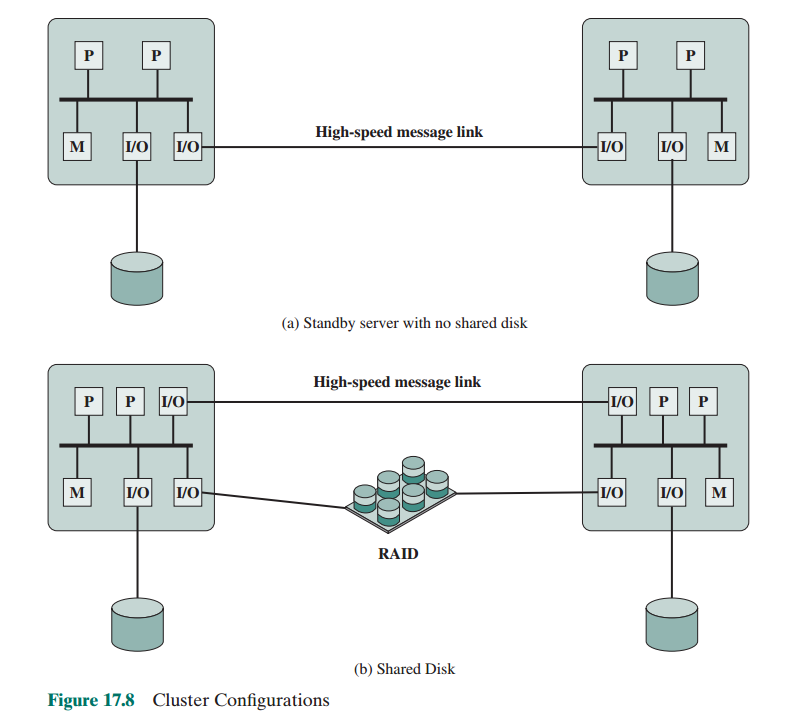
［BREW97］列出了集群化能提供的4个好处，它们也可看成是集群设计的要求或目标：

* **绝对的可扩展性**（absolute scalability）：组建一个大的集群系统，使它的能力远超过甚至最大的独立计算机是完全可能的。一个集群系统能有上百甚至儿千台机器。每台机器是一个多处理器。
* **增量的可扩展性**（incremental scalability）：集群系统能以这样的方式来配置：以少量增加方式把新结点添加到集群系统中，于是，用户开始只需要一个适度规模的系统，随着需求的增长再扩展它，而无需采用以大系统替代原小系统的主要升级方式。
* **高可用性**（high availability）：因为集群系统的每个结点都是一台独立计算机，因此一个结点的故障不意味着业务的丢失。多数产品中，容错是由软件自动完成的。
* **优异的价格/性能比**（superior price/performance）：通过将商售的构件组合在一起构成一个集群系统，其计算能力等于或大于单一大型机器，而其成本却低得多。

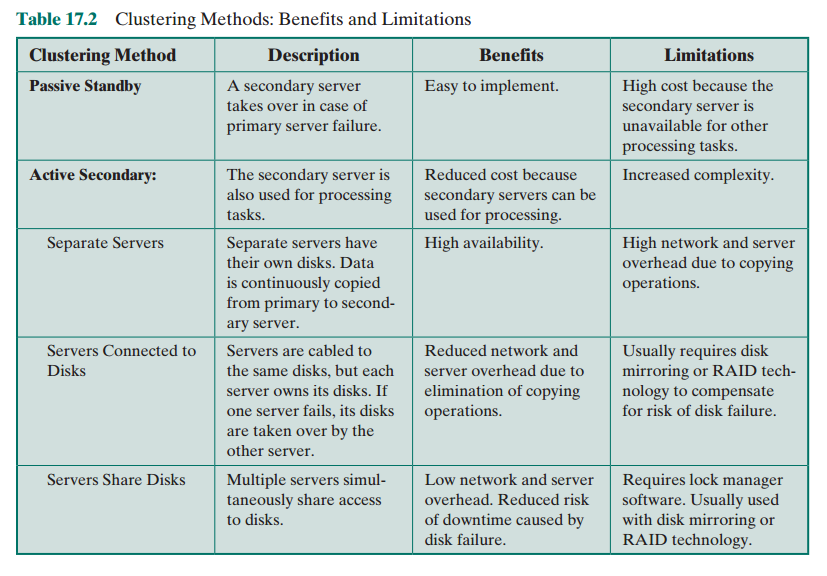
### 集群配置

在文献中，集群系统有几种分类方式。也许最简单的分类法是，根据集群系统中的计算机是否共享对同一磁盘的存取来分类。图17-8a表示了一个两结点的集群系统，高速链路是两结点的唯一互连机制，它能用于高速消息交换，以协调系统的动作。此链路可以是与其他非集群系统计算机共享的LAN，也可以是专用的互连设施。在后一种情况下，集群系统中的一个或多个计算机将有一个到LAN或WAN的链路，这样就提供了服务器集群系统与远程客户系统之间的连接。注意，图中每台计算机被描述为一台多处理器，实际上不是必须如此，但这样做会增强性能和可用性。

在图17-8描述的简单分类中，另一种方法是共享磁盘的集群系统。这种情况下，通常在结点间仍有一条消息链路；此外，还有一个直接连接到系统内多台计算机的磁盘子系统。图中的公共磁盘子系统是一个RAID系统。使用RAID或某种类型的冗余磁盘技术，在集群系统中是很普遍的，这样可使多台计算机所实现的髙可用性不被作为单故障点的共享磁盘所损害。



考察功能的可替换性可得到集群系统更清楚的分类图谱。表17-2提供了一个沿功能主线的  
有效分类法，下面就对它进行讨论。



一种通用的老式方法称为**被动式备用**（**passive standby**），其原理很简单，一台计算机处理所有工作，而另一台计算机不工作，但时刻准备着在主服务器出故障时去接替工作。为协调机器动作，活动的主服务器要周期地发送“心跳”（heartbeat）消息给备用服务器。一旦这些消息停止到达，备用服务器将认为主服务器失败了，并立即投入运行。这种方法提高了可用性，但没有改善性能。而且，两者之间交换的信息只是心跳信息，并且二者不共享磁盘，即备用者提供了功能上的备份，但不能存取主服务器管理的数据库。

这种被动式备用通常不被看成是集群系统。术语“集群系统”是专门用于这样的多个计算机互连的系统：每个计算机都是主动地进行处理，而对外维护了一个单一系统映像。术语“主动式辅助” （*active secondary*）经常用于指这种配置，它又分成3种集群化方式，分别标识为：分立的服务器、无共享和共享存储。

一种集群化方式是，每个计算机是带有自己磁盘的**分立的服务器**（**separate server**），计算机之间无共享磁盘（见图17-8a），这种安排提供了高性能以及高可用性。这种情况下，需要某种管理或调度软件，以将到来的客户请求指派给适当的服务器，而实现负载平衡和高利用率。同时要求系统具有故障接管能力，这意味着一台计算机在执行应用程序期间出现故障，另一台计算机能接替它并完成此应用服务。因此，数据必须在计算机之间连续不断地被复制，以使每台计算机都能存取其他计算机的当前数据。这种数据交换开销保证了高可用性，却付出了性能损失的代价。

为减少通信开销，现在大多数集群系统由连接到公共磁盘的服务器组成（见图17-8b）。这种方法的另外说法简称为“**无共享**”（**shared nothing**），集群的公共磁盘划分成卷，每卷被单一计算机所拥有。如果某台计算机出了故障，则集群系统必须重新配置，以使另外某台计算机拥有失效计算机的卷。

使多台计算机同时共享同一磁盘也是可能的，这称为“**共享磁盘**”（**shared disk**）方法，这样每台计算机可存取所有磁盘上的所有卷。这种方法要求使用某种类型的锁机制。以保证数据一次仅能被一台计算机存取。

### 操作系统设计问题

全面发掘集群系统的硬件潜力，要求对单机的操作系统进行某些增强。

1. **故障管理（failure management）**

集群系统如何管理故障，取决于所采用的集群化方法（见表17-2）。通常，对待故障有两种方法：高可用的集群系统和容错集群系统。高可用的集群系统呈现出所有资源都正在使用的高概率。如果出现像系统停机或磁盘卷丢失这样的故障，那么正在进行的查询可能丢失。如果进行重试，丢失的查询将由集群系统中另一台计算机提供服务。然而，集群操作系统并不保证部分已执行但未完成事务的状态，这需要应用级处理。

容错的集群系统保证所有资源总是可用的。这是通过使用冗余共享磁盘技术、回退未提交事务、提交已完成事务的机制来实现的。

应用程序和数据资源从一台失效计算机切换到集群系统中的另一台计算机上，称为**故障接管**（**failove**r）。一旦故障被修复了，应用程序和数据恢复到原计算机，称为**故障退回**（**failback**）。故障退回能自动进行，但这要求故障是真正修复了，并且不会再发生；如果不是这样，那么自动故障退回会造成随后各计算机随机发生资源故障，导致性能下降和难以恢复的问题。

1. **负载平衡（load balancing）**

集群系统要求一种在可用计算机之间有效平衡负载的能力，对增量式扩展的集群系统也要求具备此能力。当一台新计算机加入到集群系统中，负载平衡机制应该自动将这台计算机列入调度应用程序可使用的资源。中间件机构需要识别集群系统不同成员上的服务，并可将服务由一个成员迁移到另一个成员上。

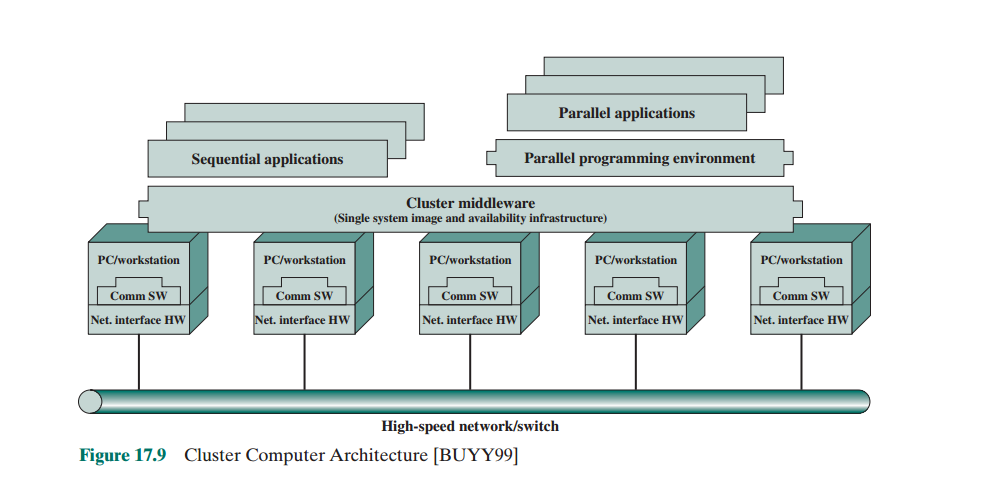
1. **并行化计算（paralleling computation）**

某些情况下，集群的有效利用要求并行执行来自单个应用的软件。［KAPPOO］对此问题列出了三种通常方法：

* **并行化编译器（parallelizing compiler）：**并行化编译器在编译时确定应用的哪些部分可并行执行。然后，将它们离析出来指派给集群中不同的计算机。性能取决于问题的性质和编译器设计得是否恰当。
* **并行式应用（parallelized application）：**在这种方法中，程序员从一开始就按照程序在集群上运行的设想来编写程序，并使用消息传递方式在各结点间传送所要求的数据。这对于程序员来说，负担不轻；对于某些应用来说，却是利用集群能力的最好办法。
* **参数式计算（parametric computing）：**如果应用的实质是一个算法或程序的多次执行，每次有一组不同的起始条件或参数，那么可以使用这种方法。一个典型的例子是模型模拟，它将多次运行不同的模拟方案，然后再对结果进行分析。为有效地使用参数式计算方法，需要使用参数处理工具软件以便有序地组织、运行和管理作业。

### 集群计算机体系结构

图17-9给出一个典型的集群体系结构。各个计算机通过高速LAN或开关硬件互连，每个计算机都能独立运行。各计算机都安装中间件（middleware）软件层，以允许集群操作。集群的中间件对用户提供了一个统一的系统映像，称为单一系统映像（single-system image）；它还通过负载平衡和响应个别部件故障的方式，来负责提供高可用性。［HWAN99］列出如下作为集群中间件的服务和功能要求：



* **单一入口点（single entry point）：**用户登录到集群上而不是登录到个别计算机上。
* **单一文件层次（single file hierarchy）：**用户看到的是，同一根目录下的单一文件目录层次体系。
* **单一控制点（single control point）：**默认的工作站用于整个集群的管理和控制。
* **单一虚拟网络（single virtual networking）：**尽管实际的集群配置可能由多个互连的网络组成，但任一结点都能访问集群中任何其他结点。这形成单一虚拟网络的操作。
* **单一存储空间（single memory space）：**分布式共享存储器允许程序共享变量。
* **单一作业管理系统（single job-management system）：**在集群作业调度程序管理下，用户提交作业无需指定执行此作业的宿主计算机。
* **单一用户接口（ single user interface）：** 一个公共图形接口支持所有用户，不管用户由哪台工作站进入集群。
* **单一I/O空间（single I/O space）：**任一结点都能远程访问任何I/O外围或磁盘设备，而无需知晓它们的物理位置。
* **单一进程空间（single process space）：**使用一致的进程标识方式，任何结点上的进程都能在远程结点上生成另一进程并与之通信。
* **检查点（checkpointing）：**这个功能周期性地保存进程状态和中间计算结果，以便在故障修复之后重新运算。
* **进程迁移（ process migration）：**这个功能可使负载平衡。

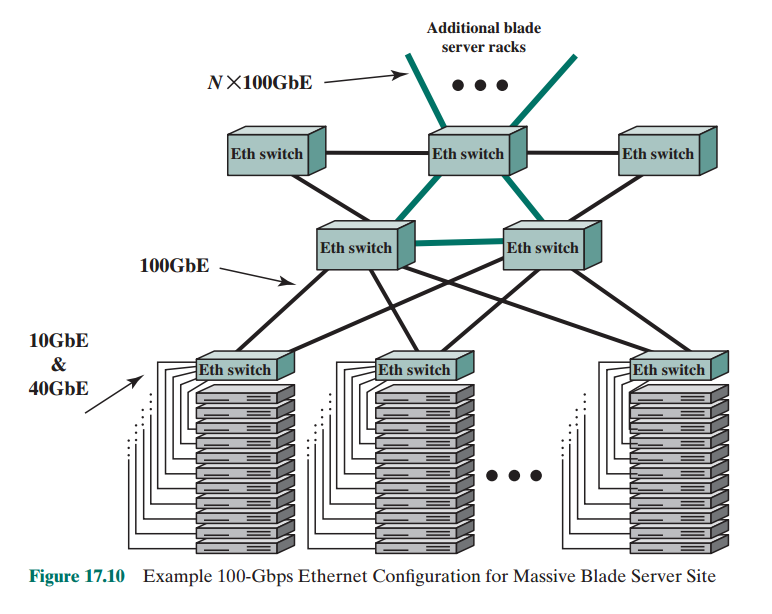
上述的最后4项用于提高集群的可用性。其余所有项都涉及提供单一系统映像。

再返回到图17-9，集群还要包括软件工具，已使并行程序能有效地运行。

### 刀片服务器

实现集群的一种常用方式是刀片服务器。刀片服务是一种服务器体系结构，它把多个服务器模块（刀片）装入到单个机箱中。刀片服务器广泛应用于数据中心，以便节省空间，并提高系统管理效率。刀片服务器可能采用塔式或机架式安装，无论哪一种，电源都是由机箱统一提供。机箱内每个“刀片”包含它自己的处理器、内存和硬盘。

　　刀片服务器的一个应用例子如图17-10所示。大型数据中心通常装备着大量的刀片服务器，其发展趋势是给各个服务器装配10Gb/s的网络接口，以便处理这些服务器上大规模多媒体数据交换。这样的配置给连接这些众多服务器的以太网交换机带来了巨大的压力。100Gb/s的带宽能提供处理高速增长的数据交换要求的能力。于是，100Gb/s的以太网交换机就被安装到数据中心的上行交换链路（switch uplink）上，以提供楼宇之间、园区之间以及广域的企业网络连接。



### 集群与SMP的对比

集群系统和对称多处理器都提供了一种使用多个处理器来支持高需求应用的配置，二者都已有商业化产品可用，虽然SMP产品出现要早得多。

SMP方法的主要优势在于，SMP比集群系统更容易配置和管理。SMP更接近原来的单处理器模型，而几乎所有的应用都是以此模型来编写的。从单处理器到SMP主要的变化在于调度程序的功能增强。 SMP的另一优点是，与具有可比性的集群系统相比，SMP占用较少的物理空间，并且耗电较少。另一重要优点是，SMP产品已很好地建立起来了，并且很稳定。

然而，经过长时间的竞争后，集群系统方法的优势将使它占领高性能服务器市场。在增量和绝对可扩展性方面，集群系统的优势是无可匹敌的。同时，在可用性方面，集群系统也是大大强于SMP的，因为集群系统中的所有部件都能很容易地做到高冗余度。

## 17.6非均匀存储器访问

以商售产品形式提供多处理器系统的两种通常方法是SMP和集群系统。近年来，一种被称为非均匀存储器访问（NUMA）的方法已成为研究的热点，并且NUMA商业产品最近也已问世。

开始介绍之前，先定义经常出现在NUMA文档中的一些术语。

* **均匀存储器访问（uniform memory access， UMA）：**所有处理器都可使用装载和保存指令存取主存的所有部分。一个处理器对所有存储区域的访问时间是相同的，不同处理器所进行的存储器访问时间也是相同的。17.2节和17.3节所讨论的SMP组织就是一种 UMA。
* **非均匀存储器访问（nonuniform memory access，NUMA ）：**所有处理器使用装载和保存指令能访问所有主存部分。根据正被访问的存储器区域。一个处理器所用的存储器访问时间是不同的。这种情况对所有处理器也是真的；然而，哪个区域快些，哪个区域慢些，对不同处理器是不同的。
* **cache一致的 NUMA （cache-coherent NUMA， CC-NUMA）：**在各处理器的 cache 之间有维护cache一致性机制的NUMA系统。

一个无cache 一致性维护的NUMA系统或多或少等同于一个集群系统。令人关注的商业产品是CC-NUMA系统，它与SMP和集群系统有着明显区別。通常但不总是，这样的系统在商业文档中实际被称为CC-NLMA系统。本节也仅关注CC-NUMA系统。

### 动机

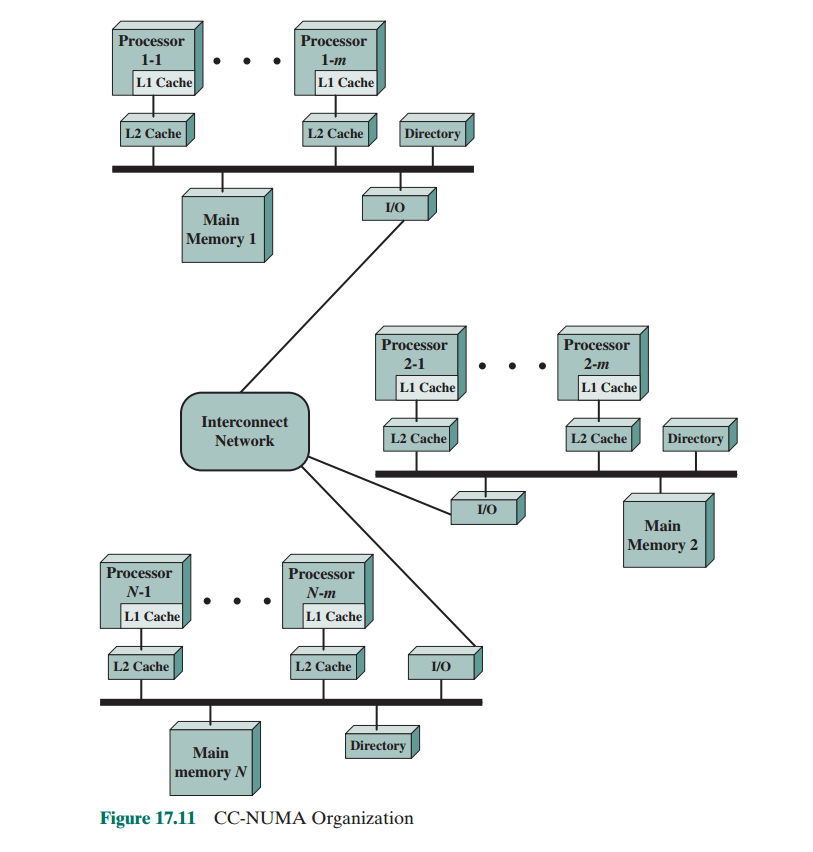
在SMP系统中，可以使用的处理器数目是有实际限制的。有效的cache策略能减少任一处理器与主存之间的总线流通量。随着处理器数目的增长，这个总线流通量也增长。还有，总线被用于传递cache一致性信号，进一步增加了总线的负担。到某一点，总线变成一个性能瓶颈。性能的降低限制了 SMP中可配置的处理器数目 大致在16 ~64之间。例如，Silicon Graphics公司的Power Challenge SMP被限制为单一系统中有64个R10000处理器；超过这个数目性能将显著降低。

SMP中的处理器数目限制也是开发集群系统的一个推动因素。然而，集群系统中是每个结点有自己的私有主存；应用程序看不到大的全局存储器。实际上，集群以软件而不是硬件维护一致性。这种存储器粒度影响着性能。为实现最大性能，必须为这种环境专门定制软件。一种达到大规模多处理而又保持SMP风格的方法是NUMA。

NUMA的目标是维护一个透明的、系统范围的存储器，并准许有多个多处理器结点，每个结点有自己的总线或其他内部互连系统。

### 组织

图17-11描述了一种典型的CC-NUMA组织。这里有多个独立的结点，每个结点实际上是一个SMP组织。于是，每个结点有多个处理器和主存，每个处理器有自己的L1和L2 cache。结点是整个CC-NUMA组织的基本构造块，如每个Origin结点包括两个MIPS R10000处理器，又如每个Sequent NUMA-Q结点包括4个Pentium II处理器。结点通过某种通信设施互连，这些设施可以是开关机构、环或其他某种网络设施。



CC-NUMA系统内的每个结点都包含某些主存。然而，从处理器的角度看，这里只有每一位置都具有唯一系统范围地址的单一可寻址的存储器。当一个处理器启动一个存储器访问时，如果所请求的存储位置不在此处理器的cache中，那么L2 cache启动一个取操作。如果所请求的行在主存的本地部分中，经由本地总线把此行取来。如果所请求的行在远程的主存区域中，那么一个请求会由cache自动发出，以便通过互联网络从远端获取该行，然后将该行放到内部总线上，并且由发送请求的从内部总线上读取。所有这些动作是自动进行的，对处理器及其cache是透明的

在这种配置中，cache一致性是主要考虑的问题。虽然实现在细节上有所不同，但一般情况下我们可认为每个结点必须维护某种类型的目录，它给出各存储器部分的位置指示和cache状态信息。为看清这种策略如何起作用，我们给出一个取自［PF1S98］的例子。假设，结点2上的处理器3 （P2-3）要求访问存储器位置798，而此位置在结点1的存储器内，那么会发生如下序列的操作：

1. P2-3在结点2的监听总线发出一个对位置798的读请求。
2. 结点2上的目录看见这个请求，并识别出此位置在结点1中。
3. 结点2的目录把一个请求发送到结点1。结点1的目录收到此请求。
4. 结点1的目录，起着P2-3代理的作用，请求读取798的内容，就像它是一个处理器一样。
5. 结点1的主存响应此请求，将所要的数据放到总线上。
6. 结点1的目录由总线读取数据。
7. 数据被传回到结点2的目录。
8. 结点2的目录将数据放回到总线，它起着代理保存此数据的原存储器的作用。
9. 数据被读取并放入P2-3的cache中，然后递交给P2-3。

上述顺序说明了如何由远程存储器读数据。它使用了硬件机制，从而使事务过程对处理器是透明的。在这种机制的背后，需要有某种类型的cache一致性协议。各种系统具体如何实现cache 一致性协议是不同的。这里我们只进行简要的一般性讨论。首先，作为上述序列操作的一部分，结点1的目录应保持一个记录，记录着某些远程cache保存着位置798所在的行。其次，需要一种协同操作的协议来处理可能发生的修改。例如，一个cache中发生了修改，这个事实能广播到其他cache。各个结点的目录收到这一广播后，确定本结点的各局部cache中是否有此行的副本，如果有，就放弃它。此修改的实际存储位置所在的结点收到此广播通知后，该结点目录维护的一个记录项指示此存储器行已无效，并一直保持下去，直到回写发生。如果任何其他处理器（本地或远程的）请求此无效行，则本地目录必须强迫发生一个回写过程，修改存储器之后，再提供此数据。

### NUMA的优缺点

CC-NUMA系统的主要优点是，它能在比SMP更高的应用级别上提供有效性能，而不要求软件进行大的修改。由于有多个NUMA结点，因此任何个别结点上的总线流通量被限制到总线能处理的负载程度上。然而，如果多数存储器访问是对远程结点的，性能就开始变差。有两个理由可相信这种性能变差可以避免。第一，L1和L2 cache的使用被设计成减少所有的存储器访问，包括远程的访问。如果大多数的软件都有好的时间局部性，则远程存储器访问应该不会过多。第二，如果使用了虚拟存储器，并且软件有好的空间局部性，则应用所需的数据将驻留在经常使用的有限几页上；那么，这些页可以初始装入到运行应用程序所在结点的本地存储器上。 Sequent研究人员报告，这种空间局部性出现在代表性应用中［LOVE96］。最后，通过在操作系统中包括页移植机制，能增强虚拟存储器的能力，这种机制将虚拟存储页迁移到频繁使用此页的结点上。 Silicon Graphics公司的设计人员报告了这种方法的成功应用［WHIT97］。

虽然远程访问导致的性能下降可通过上述方法弥补，文献［PFIS98］中讨论了其中两个缺点。第一点，CC-NUMA不像SMP那样透明，操作系统和应用由SMP移植到CC-NUMA系统上，将要求软件做些改变。这包括上面已提到过的页分配、进程分配相由操作系统完成的负载平衡。第二点与可用性有关，这是一个相当复杂的问题，并取决于CC-NUMA系统的确切实现。有兴趣的读者请参阅［PFIS98］。

## 17.7云计算

第一章介绍了云计算，讨论了三种服务模型。这里我们更详细地说。

### 云计算元素

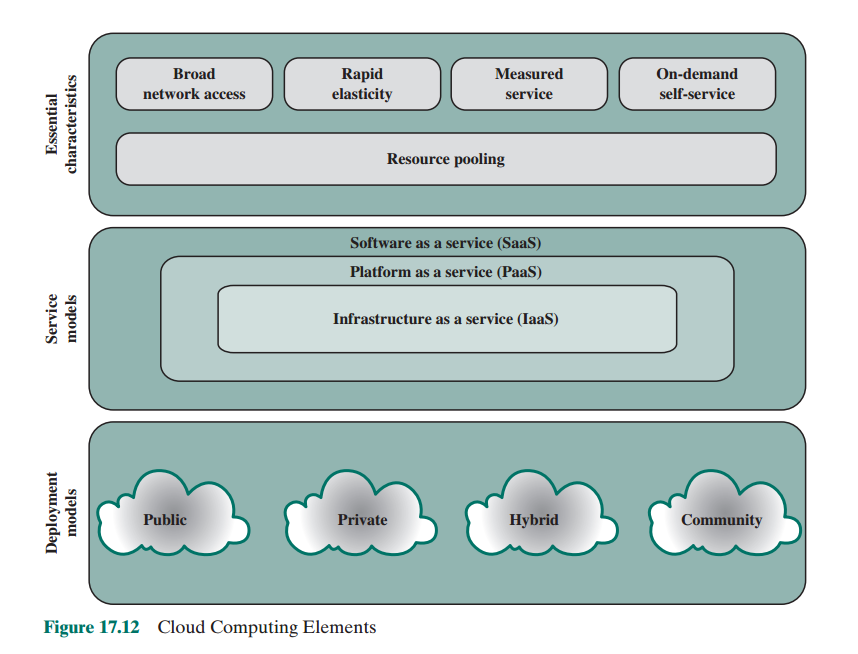
NIST SP-800-145（NIST云计算定义）规定云计算由五个基本特征、三个服务模型以及四种部署模型。图17.12说明了这些概念之间的关系。云计算的基本特征包括：

■广域网接入：能力可以通过网络获得，并且通过标准机制获得，这些标准机制促进异构的瘦或厚的客户端平台（例如，移动电话、笔记本电脑和平板电脑）的使用。作为其他传统的或基于云计算的软件服务。

■快速弹性：云计算使您能够根据特定的服务需求扩展和减少资源。例如，在特定任务期间，您可能需要大量的服务器资源。然后，您可以在完成任务后释放这些资源。

■测量服务：云系统通过利用适合于服务类型（例如，存储、处理、带宽和活动用户账户）的某种抽象级别的计量能力来自动控制和优化资源使用。资源使用可以被监视、控制和报告，为所使用的服务的提供者和消费者提供透明度。

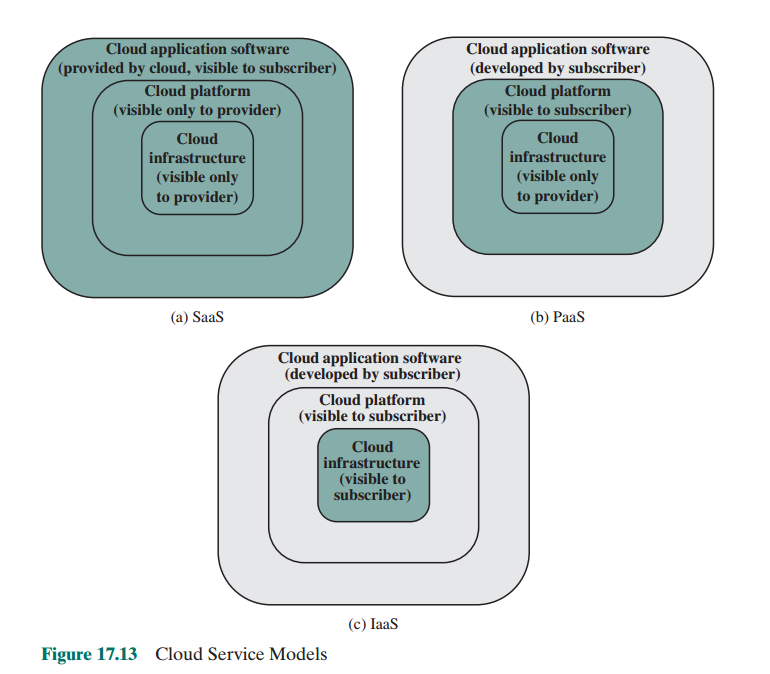
 ■按需自助服务：用户可以根据需要自动单方面提供计算能力，如服务器时间和网络存储不需要与每个服务提供者进行人工交互。因为服务是按需提供的，所以资源不是IT基础设施的永久部分。



■资源池：使用多租户模型将提供者的计算资源汇集为多个消费者服务，根据消费者需求动态地分配和重新分配不同的物理和虚拟资源。存在某种程度的位置独立性，因为客户总体上对所提供的资源的确切位置没有控制或知识，但是可能能够在更高抽象级别（例如，国家、州或数据中心）指定位置。资源的示例包括存储、处理、内存、网络带宽和虚拟机。甚至私有云也倾向于在同一组织的不同部分之间汇集资源。

NIST定义了三个服务模型，它们可以被看作嵌套服务替代（图17.13）。这些定义在第1章，可以简要概述如下：

■软件即服务（SaaS）：以软件形式向客户提供服务，特别是应用软件，运行在云中并可访问。

 ■平台即服务（PaaS）：以平台的形式向客户提供服务，客户应用程序可以在平台上运行。

■基础设施即服务（IaaS）：为客户提供对底层云基础设施的访问。

NIST定义了四种部署模型：

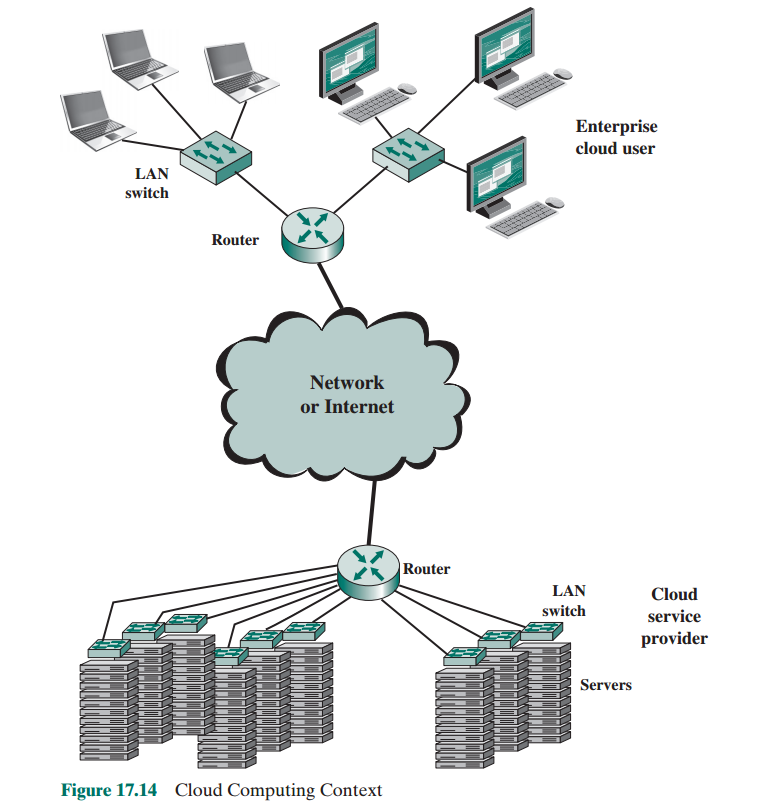
■公共云：云基础设施可向公众或大型产业集团提供，并由销售云服务的组织拥有。云提供商负责云基础设施以及云中的数据和操作的控制。公有云的主要优点是成本。订阅组织仅为其需要的服务和资源付费，并且可以根据需要调整这些服务和资源。此外，用户大大减少了管理开销。主要关心的是安全问题。然而，有许多公共云提供商已经展示了强大的安全控制，事实上，这些提供商可能拥有更多的资源和专门知识来致力于私有云中可用的安全。

■私有云：私有云是在组织的内部IT环境中实现的云基础设施。组织可以选择管理内部云或者将管理功能承包给第三方。此外，云服务器和存储设备可能预先存在或非预先存在。选择私有云的一个关键动机是安全性。私有云基础设施对数据存储的地理位置和安全的其他方面提供了更严格的控制。

■社区云：社区云共享私有云和公共云的特征。就像私有云一样，社区云不向任何订阅者开放。像公共云一样，云资源在许多独立组织之间共享。共享社区云的组织具有相似的需求，并且通常需要彼此交换数据。采用社区云概念的行业的一个例子是卫生保健行业。可以实现社区云以符合政府隐私和其他规定。社区参与者可以以受控的方式交换数据。云基础结构可以由参与组织或第三方管理，并且可以在内部或外部存在。在这个部署模型中，比起公共云（但是比私有云），成本分散在用户身上，因此仅实现了云计算的一些成本节约潜力。

■混合云：云基础设施由两个或更多个云（私有、社区或公共）组成，这些云仍然是唯一的实体，但是通过标准化或专有技术绑定在一起，这些技术支持数据和应用程序移植能力（例如，为了负载平衡而采用云突发）。云间的）使用混合云解决方案，敏感信息可以放置在云的私有区域中，而敏感度较低的数据可以利用公共云的成本效益。

图17.14说明了典型的云服务上下文。企业主要维护企业LAN或一组LAN内的工作站，这些工作站由路由器通过网络或因特网连接到云服务提供商。云服务提供商维护大量的服务器，它使用各种各样的服务器来管理这些服务器网络管理、冗余和安全工具。在图中，云基础结构显示为刀片服务器的集合，这是一个常见的体系结构。



### 云计算参考架构

NIST SP 500-292（NIST云计算参考体系结构）建立了参考体系结构，描述如下：

NIST云计算参考体系结构关注于“什么”云服务提供的需求，而不是“如何”设计解决方案和实现。参考体系结构旨在帮助理解云计算中的操作复杂性。它不表示特定云计算系统的系统体系结构；相反，它是使用公共参考框架描述、讨论和开发特定于系统的体系结构的工具。

NIST基于以下目标开发了参考体系结构：

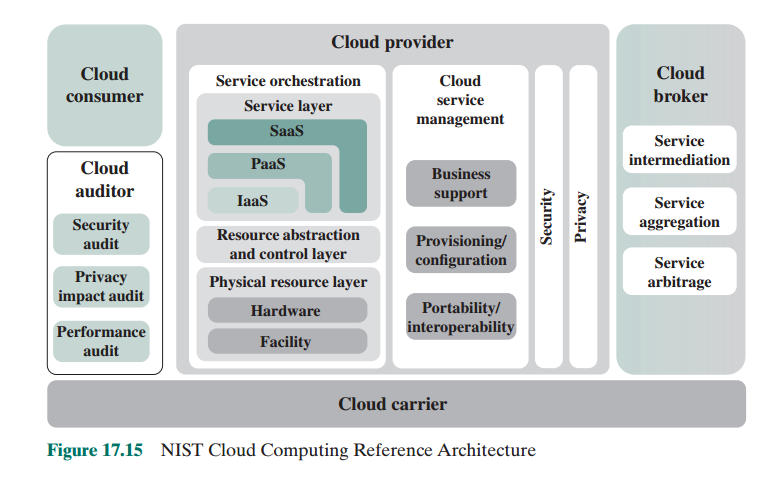
■在整体云计算概念模型的上下文中说明和理解各种云服务。

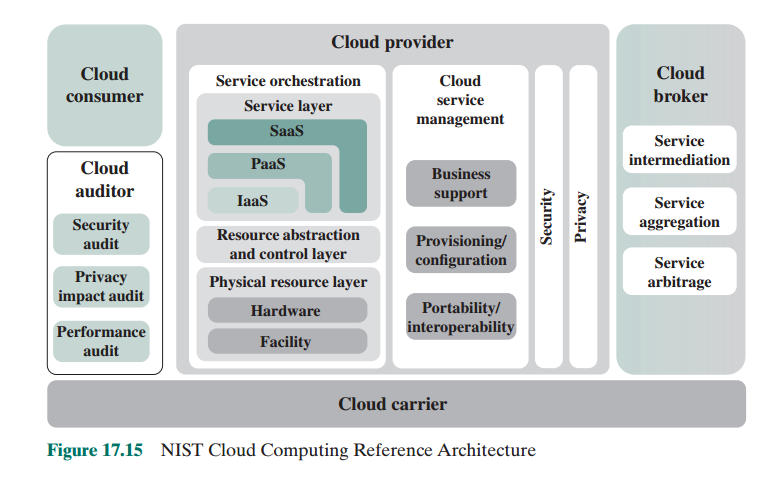
■为消费者理解、讨论、分类和比较云服务提供技术参考。

■便于分析安全性、互操作性、可移植性和参考实现的候选标准。

图17.15所示的参考体系结构根据角色和责任定义了五个主要参与者：

■云消费者：与云提供商保持业务关系并使用云提供商的服务的个人或组织。





■云提供商（CP）：负责向相关方提供服务的人、组织或实体。

■云审计：能够对云服务、信息系统操作、性能和云实现的安全性进行独立评估的一方。

■云代理：管理云服务的使用、性能和提供，并协商CP和云消费者之间关系的实体。

■云载体：一种中介，提供从CP到云消费者的云服务的连接和传输。

云消费者和提供者的角色已经讨论过了。总之，云提供商可以提供一个或多个云服务以满足云消费者的IT和业务需求。对于三个服务模型（SaaS、PaaS、IaaS）中的每一个，CP提供支持该服务模型所需的存储和处理设备，以及云服务消费者的云接口。对于SaaS，CP在云基础设施上部署、配置、维护和更新软件应用程序的操作，以便以预期的服务级别向云消费者提供服务。SaaS的消费者可以是向其成员提供对软件应用程序的访问的组织、直接使用软件应用程序的最终用户或为最终用户配置应用程序的软件应用程序管理员。

对于PaaS，CP管理平台的计算基础结构，并运行提供平台的组件的云软件，例如运行时软件执行堆栈、数据库和其他中间件组件。PaaS的云消费者可以使用CP提供的工具和执行资源来开发、测试、部署和管理托管在云环境中的应用程序。

对于IaaS，CP获取服务底层的物理计算资源，包括服务器、网络、存储和托管基础设施。IaaS云消费者反过来使用这些计算资源，例如虚拟计算机，以满足其基本的计算需求。

云运营商是一个网络设施，它提供云消费者和CP之间的云服务的连接和传输。通常，CP将与云运营商建立服务水平协议（SLA），以提供与提供给云消费者的SLA水平一致的服务，并且可能要求云运营商在云消费者和CP之间提供专用的和安全的连接。

云代理在云服务太复杂而无法由云消费者轻松管理的情况下非常有用。云中介可以提供三个方面的支持：

■服务中介：这些是增值服务，例如身份管理、性能报告和增强的安全性。

■服务聚合：代理组合多个云服务以满足单个CP没有具体解决的消费者需求，或者优化性能或最小化成本。

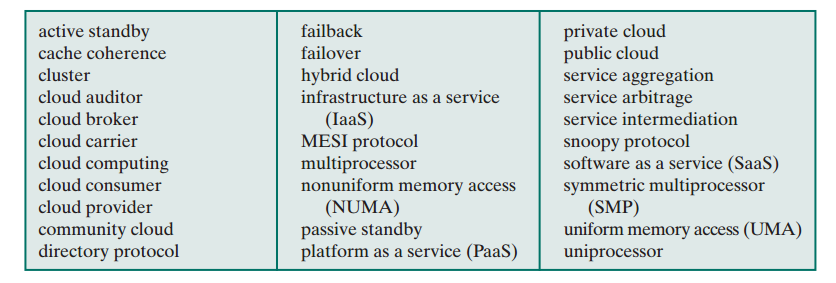
■服务套利：这与服务聚合类似，只是正在聚合的服务没有固定。服务套利是指经纪人具有从多个代理中选择服务的灵活性。云经纪人

例如，可以使用信用评分服务来衡量和选择具有最佳评分的机构。

云审计师可以根据安全性控制、隐私影响、性能等对CP提供的服务进行评估。审计师是一个独立的实体，可以确保CP符合一组标准。

## 17.8关键术语、思考题和习题

### 关键术语



### 思考题

17.1 列出并简要定义三算机系统组织的三种类型。

17.2 SMP的主要特征是什么？

17.3 与单处理器相比，SMP有哪些潜在优势？

17.4 公共汽车组织的优点和局限性是什么？

17.5 软件和硬件的cache一致性方案有何不同？

17.6 MESI协议有4种状态，每种状态的意思是什么?

17.7 集群化的主要好处是什么？

17.8 故障转移和故障退回有何不同？

17.9 UMA、NUMA和CC-NUMA三者之间的区别是什么？

17.10 云中介可以提供什么支持？

### 习题

17.1假定一个计算机系统中有*n*个处理器，每个处理器的执行速率是*x* MIPS，程序代码有α比例部分能在*n*个处理器上同时执行，其余部分必须在单处理器上顺序执行。

（a）请给出程序在这一系统上运行时的有效执行速率，用*n*、*x*和α表示。

（b）若*n*＝16， *x*＝4MIPS，则α应为何值才能产生40MIPS的系统性能。

17.2 一台有8个处理器的多处理器连接20台磁带机。有大量的作业提交给系统，每个作业最大要求4台磁带机才能完成执行。假定每个作业开始运行时只需要3台磁带机，长时间运行之后到结束前才短时间需要第4台磁带机，并假设这些作业是循环不断地被提交的。

（a）如果OS中的调度程序是这样工作：没有4台磁带机可用，不启动作业；一旦启动作业，4台磁带机立即被指派，直到作业结束才释放。问：一次能进行的最大作业数是多少？作为这种调度策略的结果，处于空闲的磁带机最大数和最小数各是多少？

（b）请推荐一种策略，它能改善磁带机的利用率，而又避免系统死锁。此时，一次能进行的最大作业数是多少，空闲磁带机上限和下限又是多少？

17.3你能预见在基于总线的多处理器上采用cache写一次（write-one）方法会有什么问题吗？若有，请推荐一种解决方法。

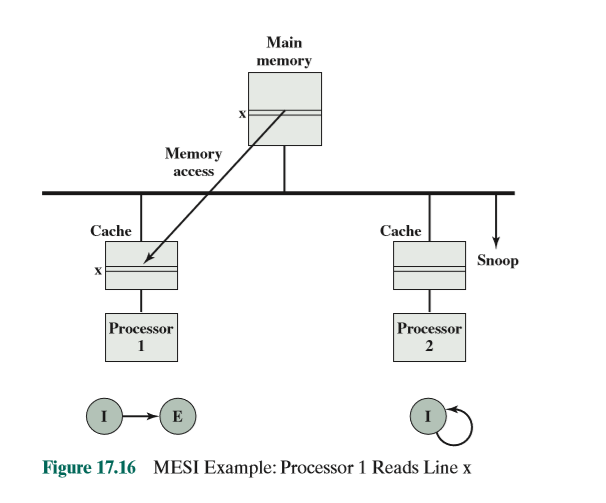
17.4考虑一个SMP配置中只有两个处理器的情况，每个处理器有一个cache，并使用MESI协议。经过一段时间后。两个处理器都要访问存储器的统一数据行*x*，并假定最初两cache中有此行的无效副本。图17-16给出处理器P1读*x*行后的结果。请以此为初始情况，画出如下操作序列的后续图：

（1）P2 读*x。*

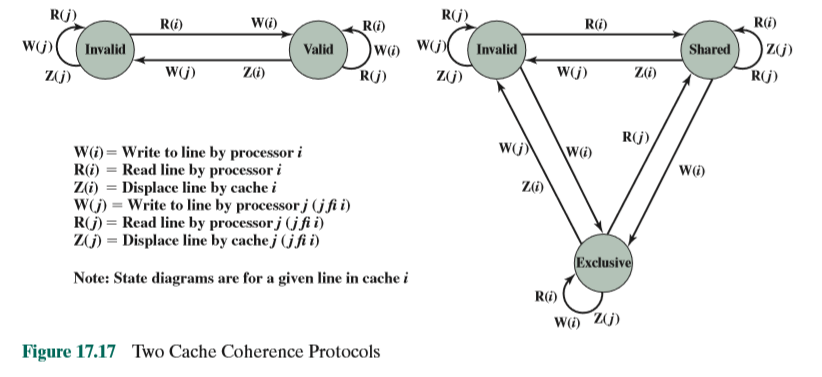
（2）P1写*x* （为清楚起见，此PI cache行标记*x’*）。

（3） P1 写*x*（此 PI cache 行标记*xn*）。

（4）P2 读*x*。

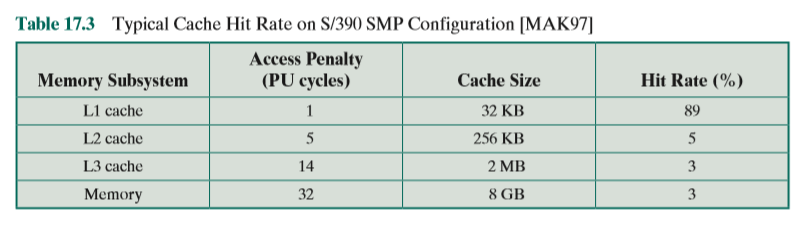


17.5图17-17表示两种可能的cache 一致性协议的状态图，请推导并说明每种协议，并将其与MESI协议做比较。



17.6 考虑一个有L1和L2 cache并使用MESI协议的SMP。正如17.3节所述，L2 cache的每行有4种可能的状态。L1 cache行是否也需要4种状态？如果不是，请说明什么状态能取消。

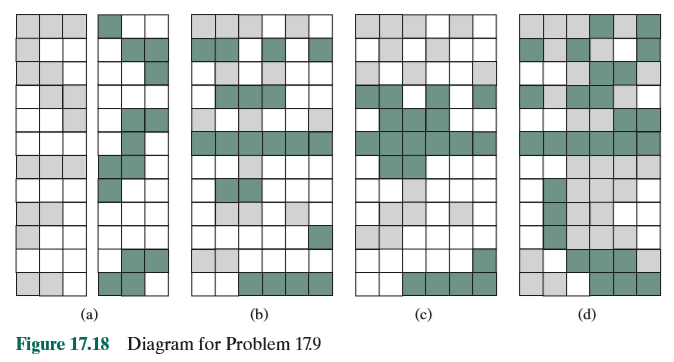
17.7 IBM大型机的一个早期版本（S/390 G4），使用了三级cache。与z990相同，仅LI cache在处理器芯片（称为处理器单元，PU）上。L2 cache类似于z990。L3 cache在一分立芯片上，位于L2 cache和存储器卡之间，起着存储控制器作用。表17-3表示了 IBM S/390三级cache配置情况下的性能。这个问题的目的在于确认包括第3级cache是否有价值。请确定只有L1 cache的系统的存取开销（平均PU周期数）；并将得到的值归一化到1.0。然后，再确定使用L1和L2 cache以及使用全部三级cache的系统存取开销。记下每种情况下的改进总量，并表明你对L3 cache价值的看法。



17.8 （a）考虑一个带有指令和数据分立cache的单处理器，其命中率分别是*H*i和*H*d。由处理器到的cache存取时间是*c*个时钟周期，存储器和cache间的块传送时间是*b*个时钟周期。令*f*i是取指占存储器访问的比例，*f*d是数据cache中脏行占被替换行的比，假设采用回写策略，请用上述定义的参数确定有效的存储器访问时间。

（b）假定有一个基于总线的SMP，其中每个处理器都有（a）部分所述的特征。除了存储器读写之外，每个处理器还必须完成each作废处理：这将影响有效的存储器访问时间。令*f*inv是引发作废信号的数据访问比例，此信号被传送到其他数据cache。发送此信号的处理器要用*t*时钟周期完成作废操作，其他处理器不参与此次作废操作。请确定有效的存储器访问时间。

17.9 图17-18分别暗示的是什么组织方案？



17.10 图17-7中，某些图表示出水平行部分填充，另一种情况是整行空白。请说明这代表哪两种效率损失。

17.11 考虑图14-13b所描述的流水。现在不管指令取指和译码两阶段，将它改画成图17-19a来代表线程A的执行。图17-19b代表另一线程B的执行。两种情况处理器使用的都是简单流水。

（a）为每个线程M出类似图17-7a的指令发射图。

（b）假定在片上多处理器上并行执行这两个线程，芯片上两个处理器核每个都使用简单流水。请画出类似图17-7k的指令发射图，并再画出有图17-19风格的流水执行图。

（c）假定有一个双发射超标量体系结构为交错式多线程超标量的实现，重复（b）问题，假定无数据相关性。注意：这里没有唯一的答案，你需对延迟和优先级进行假设。

（d）对于附塞式多线程超标量的实现，重复（c）问题。

（e）对于4发射SMT体系结构，重复（c）问题。

17.12 一个应用程序在9台计算机的集群上执行，花费了*T*时间。测试进一步表明，*T*的25%时间是此应用程序在9台计算机上同时运行，其余时间应用程序仅在单一计算机上运行。

（a）请计算，上述情况比之程序在单计算机上运行所获得的有效加速比，再计算上述程序中并行化代码（编程或编译时使用了集群模式）所占的百分比α。

（b）假设我们能有效地使用18台而不是9台计算机来运行并行化代码部分，请计算所实现的有效加速比。

17.13 在一计算机上执行的FORTRAN程序如下，其并行化版本将在一个由32台计算机组成的集群系统上运行。

L1 ： 　　 **DO** I＝1，1024

　　　L2 ：　　　　　 SOM（I）＝０

　　　L3 ：　　　　　 DO 20J＝1，Ｉ

　　　L4 ：20 SUM（I）＝SUM CI） +1

　　　L5 ：10 CONTINUE

假设行2和行4每个都用2个机器周期，包括所有的处理器和存储器访问动作。不计软件循环控制语句（行1，3， 5）引起的开销和其他所有系统开销以及资源冲突。试问：

（a）程序在单个计算机上总的执行时间是多少（以机器周期为单位）？

（b）将I循坏的选代以如下方式在32个计算机中分摊：计算机1执行最初的32次迭代（I=1到32），计算机2执行下一个32次迭代，如此继续分摊下去。总的执行时间是多少？与（a）比较，速度提高了多少？（注意，J循环支配的计算负载在各计算机中末做均衡）

（c）说明应如何修改上述并行化方式，以使计算负载在32个计算机上分布均衡地并行执行。负载均衡意味着对I、J两个循环都要将相等数量的加法运算指派到各计算机上。

（d）32台计算机并行执行所需要的敁小执行时间是多少？与单个计算机相比，速度提高多少？

17.16 考虑两向量相加的如下两个程序形式：

|  |  |
| --- | --- |
| L1：　 **DO** 10I＝１，N  L2：　 A（I）＝B（I）＋C（I）  L3：10 　**CONTINUE**  L4： SUM＝０  L5：　 **DO** 20J＝１，N  L6： SUM＝SUM＋A（J）  L7：20 **CONTINUE** | **DOALL** K＝１，M  **DO** 10 I＝L（K－1）＋1，KL  A（I）＝B（I）＋C（I）  **CONTINUE**  SUM（K）＝0  **DO** 20 J＝1，L  SUM（K）＝SUM（K）＋A（L（K-1）+J）  **CONTINUE**  **ENDALL** |

（a）左边的程序在单一处理器上执行。假设执行L2、L4和L6每行用1个处理器时钟周期。为简化，不考虑其他代码行所要求的时间。最初，所有数组已装入主存就绪，并且此短程序段已在指令cache中。执行这个程序要用多少时钟周期？

（b）右边的程序打算在一个有*M*个处理器的多处理器上执行。我们将循环操作部分划分成*M*个段，每段L = N/M个元素。DOALL声明，所有*M*个段并行执行。执行的结果是产生*M*个部分和。假定，经由共享存储器的处理器之间的通信操作每次都需要*k*个时钟周期，这样每次部分和的加法要求*k*个时钟周期。 一个*l*级二进制加法器能合并所有的部分和，这里**。 产生最终结果需要多少周期？

（c）假定数组有个元素，并且M=256。使用多处理器能实现的加速比是多少？假定*k*=200。这个加速比是因子为256的理想加速比的百分之几？