第10章 多处理机

张晨曦 刘依

www.GotoSchool.net

xzhang2000@sohu.com

10.1 引 言

1. 单处理机系统结构正在走向尽头？

2. 多处理机正起着越来越重要的作用。近几年来，人们确实开始转向了多处理机。

- Intel于2004年宣布放弃了其高性能单处理器项目，转向多核（multi-core）的研究和开发。
- IBM、SUN、AMD等公司
- 并行计算机应用软件已有了稳定的发展。
- 充分利用商品化微处理器所具有的高性能价格比的优势。

3. 本章重点：中小规模的计算机(处理器的个数＜32)

（多处理机设计的主流）

10.1.1 并行计算机系统结构的分类

1. Flynn分类法

SISD、SIMD、MISD、MIMD

2. MIMD已成为通用多处理机系统结构的选择，原因：

- MIMD具有灵活性；
- MIMD可以充分利用商品化微处理器在性能价格比方面的优势。

计算机机群系统（cluster）是一类广泛被采用的MIMD机器。

3. 根据存储器的组织结构，把现有的MIMD机器分为两类：

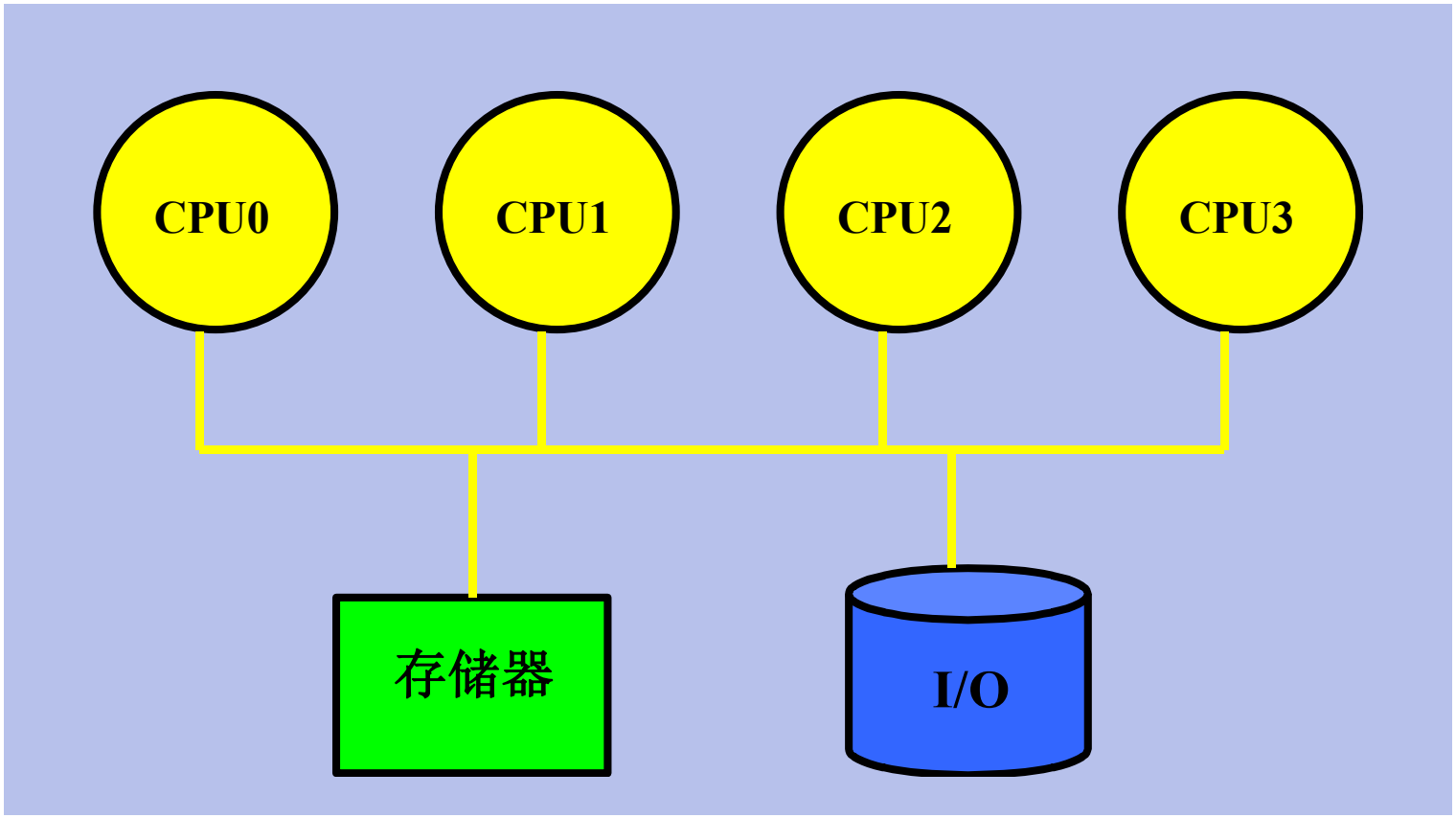
（每一类代表了一种存储器的结构和互连策略）

- 集中式共享存储器结构 动画
  - 最多由几十个处理器构成。
  - 各处理器共享一个集中式的物理存储器。

这类机器有时被称为

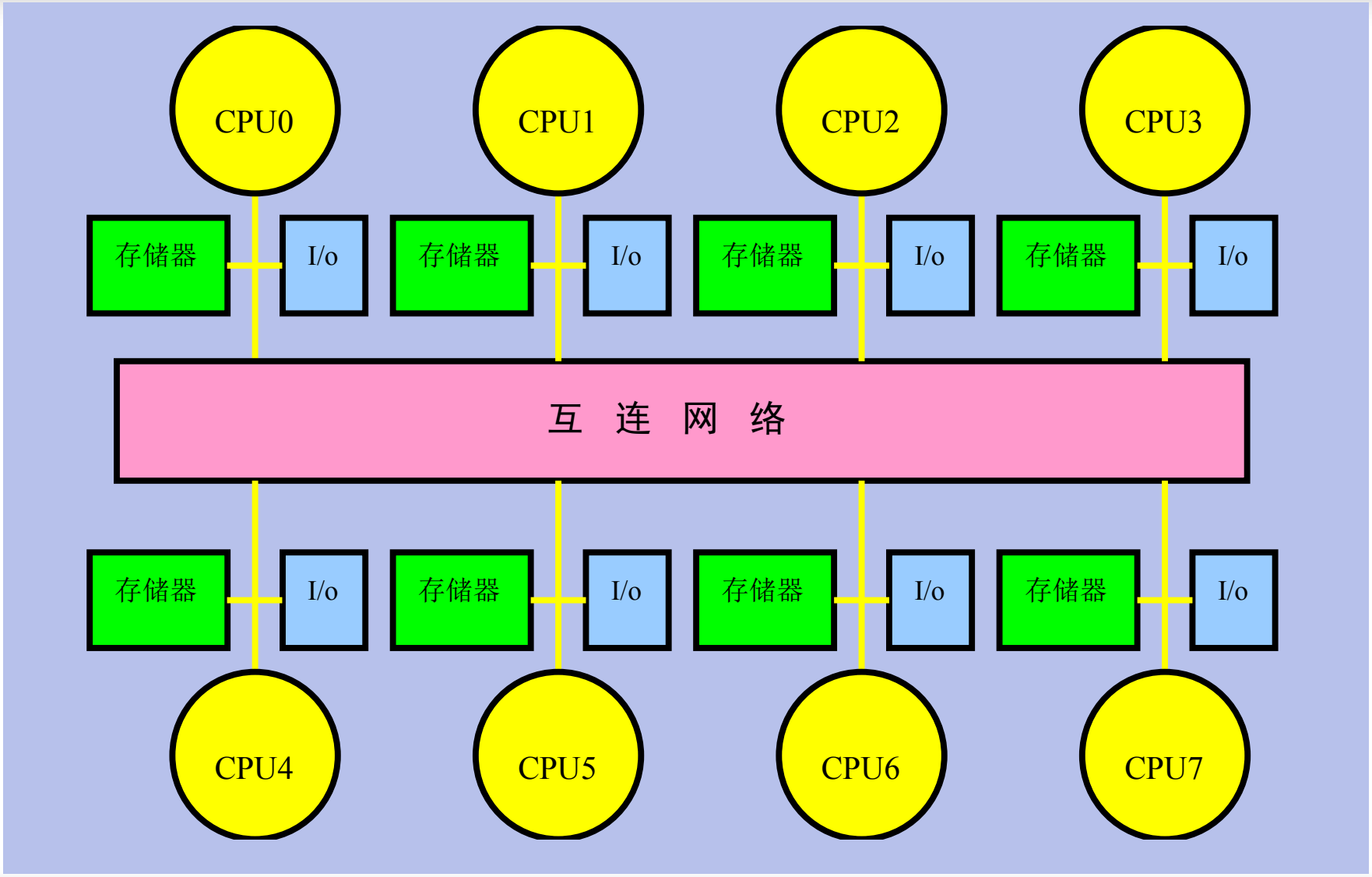
- SMP机器

  （Symmetric shared-memory MultiProcessor）
- UMA机器（Uniform Memory Access）



对称式共享存储器多处理机的基本结构

- 分布式存储器多处理机 动画
  - 存储器在物理上是分布的。
  - 每个结点包含：
    - 处理器
    - 存储器
    - I／O
    - 互连网络接口
  - 在许多情况下，分布式存储器结构优于集中式共享存储器结构。


CPU0
CPU1
CPU2
CPU3
存储器
I/o
存储器
I/o
存储器
I/o
存储器
I/o
互 连 网 络
存储器
I/o
存储器
I/o
存储器
I/o
存储器
I/o
CPU4
CPU5
CPU6
CPU7

- 将存储器分布到各结点有两个优点
  - 如果大多数的访问是针对本结点的局部存储器，则可降低对存储器和互连网络的带宽要求；
  - 对本地存储器的访问延迟时间小。
- 最主要的缺点
  - 处理器之间的通信较为复杂，且各处理器之间访问延迟较大。
- 簇：超级结点
  - 每个结点内包含个数较少（例如2～8）的处理器；
  - 处理器之间可采用另一种互连技术（例如总线）相互连接形成簇。

10.1.2 存储器系统结构和通信机制

1. 两种存储器系统结构和通信机制

- 共享地址空间
  - 物理上分离的所有存储器作为一个统一的共享逻辑空间进行编址。
  - 任何一个处理器可以访问该共享空间中的任何一个单元（如果它具有访问权），而且不同处理器上的同一个物理地址指向的是同一个存储单元。
  - 这类计算机被称为

    分布式共享存储器系统

    （DSM: Distributed Shared-Memory）

    NUMA机器 （NUMA: Non-Uniform Memory Access）

- 把每个结点中的存储器编址为一个独立的地址空间，不同结点中的地址空间之间是相互独立的。
  - 整个系统的地址空间由多个独立的地址空间构成
  - 每个结点中的存储器只能由本地的处理器进行访问，远程的处理器不能直接对其进行访问。
  - 每一个处理器-存储器模块实际上是一台单独的计算机
  - 现在的这种机器多以集群的形式存在

2. 通信机制

- 共享存储器通信机制
  - 共享地址空间的计算机系统采用

- 处理器之间是通过用load和store指令对相同存储器地址进行读/写操作来实现的。

➢ 消息传递通信机制

- 多个独立地址空间的计算机采用
- 通过处理器间显式地传递消息来完成
- 消息传递多处理机中，处理器之间是通过发送消息来进行通信的，这些消息请求进行某些操作或者传送数据。

例如：一个处理器要对远程存储器上的数据进行访问或操作：

- 发送消息，请求传递数据或对数据进行操作；

  远程进程调用(RPC， Remote Process Call)

- 目的处理器接收到消息以后，执行相应的操作或代替远程处理器进行访问，并发送一个应答消息将结果返回。

- 同步消息传递

请求处理器发送一个消息后一直要等到应答结果才继续运行。

- 异步消息传递

数据发送方知道别的处理器需要数据，通信也可以从数据发送方来开始，数据可以不经请求就直接送往数据接受方。

3. 不同通信机制的优点

- 共享存储器通信的主要优点
  - 与常用的对称式多处理机使用的通信机制兼容。
  - 易于编程，同时在简化编译器设计方面也占有优势。
  - 采用大家所熟悉的共享存储器模型开发应用程序，而把重点放到解决对性能影响较大的数据访问上。
  - 当通信数据量较小时，通信开销较低，带宽利用较好。
  - 可以通过采用Cache技术来减少远程通信的频度，减少了通信延迟以及对共享数据的访问冲突。

- 消息传递通信机制的主要优点
  - 硬件较简单。
  - 通信是显式的，因此更容易搞清楚何时发生通信以及通信开销是多少。
  - 显式通信可以让编程者重点注意并行计算的主要通信开销，使之有可能开发出结构更好、性能更高的并行程序。
  - 同步很自然地与发送消息相关联，能减少不当的同步带来错误的可能性。

- 可在支持上面任何一种通信机制的硬件模型上建立所需的通信模式平台。
  - 在共享存储器上支持消息传递相对简单。
  - 在消息传递的硬件上支持共享存储器就困难得多。所有对共享存储器的访问均要求操作系统提供地址转换和存储保护功能，即将存储器访问转换为消息的发送和接收。

10.1.3 并行处理面临的挑战

并行处理面临着两个重要的挑战

- 程序中的并行性有限
- 相对较大的通信开销

$$\text{系统加速比} = \frac{1}{(1-\text{可加速部分比例})+\dfrac{\text{可加速部分比例}}{\text{理论加速比}}}$$

1. 第一个挑战

有限的并行性使计算机要达到很高的加速比十分困难。

例10.1 假设想用100个处理器达到80的加速比，求原计算程序中串行部分最多可占多大的比例？

解 Amdahl定律为：

$$\text{加速比}=\frac{1}{\dfrac{\text{可加速部分比例}}{\text{理论加速比}}+(1-\text{可加速部分比例})}$$

$$80=\frac{1}{\dfrac{\text{并行比例}}{100}+(1-\text{并行比例})}$$

由上式可得：并行比例＝0.9975

2. 第二个挑战：多处理机中远程访问的延迟较大

- 在现有的机器中，处理器之间的数据通信大约需要50～1000个时钟周期。
- 主要取决于：

  通信机制、互连网络的种类和机器的规模

- 在几种不同的共享存储器并行计算机中远程访问一个字的典型延迟

机器	通信机制	互连网络	处理机最大数量	典型远程存储器访问时间（ns）
Sun Starfire servers	SMP	多总线	64	500
SGI Origin 3000	NUMA	胖超立方体	512	500
Cray T3E	NUMA	3维环网	2048	300
HP V series	SMP	8×8交叉开关	32	1000
HP AlphaServer GS	SMP	开关总线	32	400

例10.2 假设有一台32台处理器的多处理机，对远程存储器访问时间为200ns。除了通信以外，假设所有其它访问均命中局部存储器。当发出一个远程请求时，本处理器挂起。处理器的时钟频率为2GHz，如果指令基本的CPI为0.5（设所有访存均命中Cache），求在没有远程访问的情况下和有0.2%的指令需要远程访问的情况下，前者比后者快多少?

解 有0.2%远程访问的机器的实际CPI为：

CPI＝基本CPI＋远程访问率×远程访问开销

＝0.5＋0.2%×远程访问开销

远程访问开销为：

远程访问时间/时钟周期时间＝200ns/0.5ns＝400个时钟周期

∴ CPI＝0.5＋0.2%×400＝1.3

因此在没有远程访问的情况下的机器速度是有0.2%远程访问的机器速度的1.3/0.5=2.6倍。

- 问题的解决
  - 并行性不足： 采用并行性更好的算法
  - 远程访问延迟的降低：靠系统结构支持和编程技术

3. 在并行处理中，影响性能（负载平衡、同步和存储器访问延迟等）的关键因素常依赖于：

应用程序的高层特性

如数据的分配，并行算法的结构以及在空间和时间上对数据的访问模式等。

- 依据应用特点可把多机工作负载大致分成两类：
  - 单个程序在多处理机上的并行工作负载
  - 多个程序在多处理机上的并行工作负载

4. 并行程序的计算 / 通信比率

- 反映并行程序性能的一个重要的度量：

  计算与通信的比率

- 计算 / 通信比率随着处理数据规模的增大而增加；随着处理器数目的增加而减少。

10.2 对称式共享存储器系统结构

- 多个处理器共享一个存储器。
- 当处理机规模较小时，这种计算机十分经济。
- 近些年，能在一个单独的芯片上实现2～8个处理器核。

  例如：Sun公司 2006年 T1 8核的多处理器
- 支持对共享数据和私有数据的Cache缓存

  私有数据供一个单独的处理器使用，而共享数据则是供多个处理器使用。
- 共享数据进入Cache产生了一个新的问题

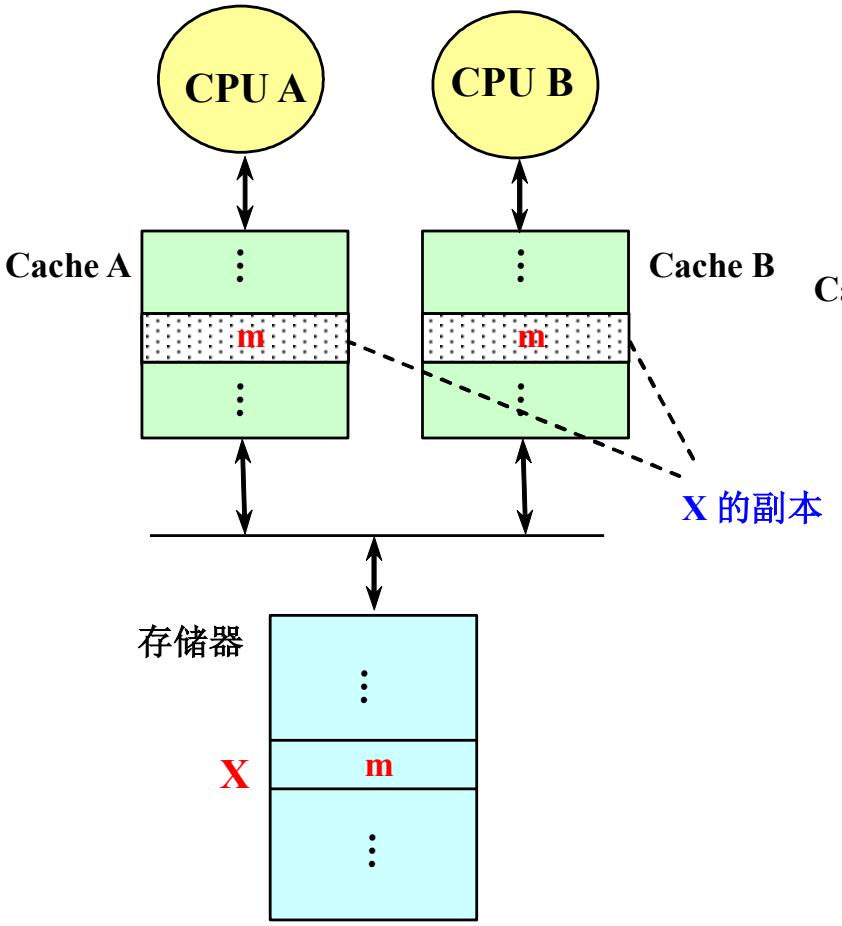
  Cache的一致性问题

10.2.1 多处理机Cache一致性

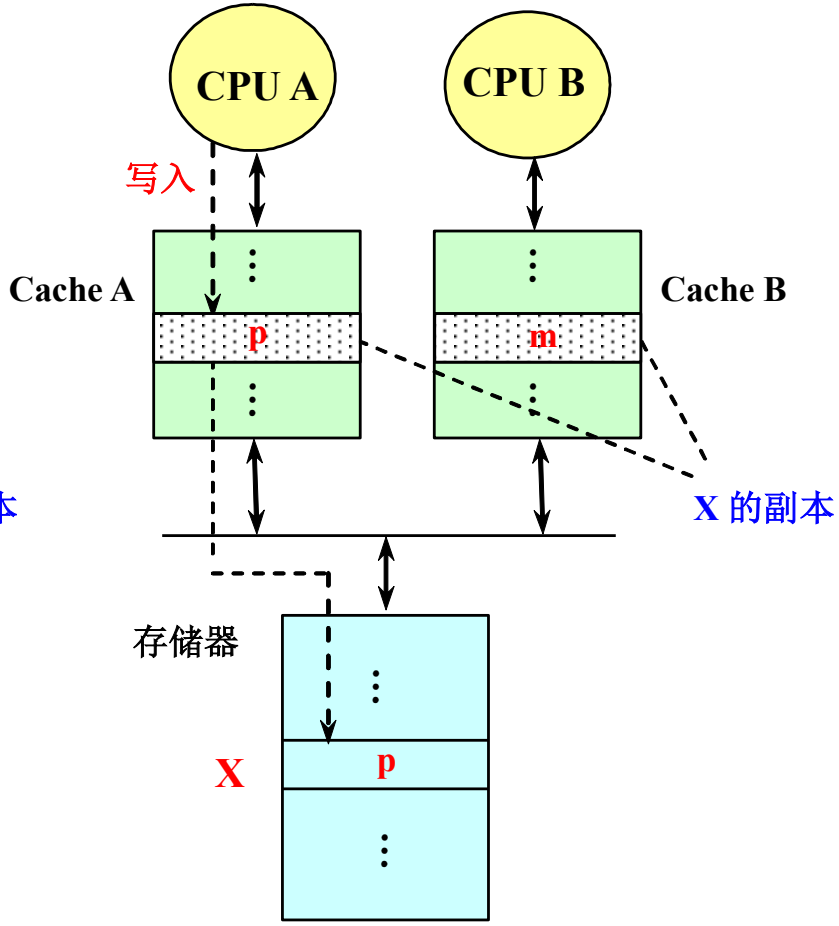
1. 多处理机的Cache一致性问题

- 允许共享数据进入Cache，就可能出现多个处理器的Cache中都有同一存储块的副本，
- 当其中某个处理器对其Cache中的数据进行修改后，就会使得其Cache中的数据与其他Cache中的数据不一致。

例 由两个处理器（A和B）读写引起的Cache一致性问题



（a） CPU A 写入前



（b） CPU A 将 p 写入 X，p≠m

2. 存储器的一致性

如果对某个数据项的任何读操作均可得到其最新写入的值，则认为这个存储系统是一致的。

- 存储系统行为的两个不同方面
  - What：读操作得到的是什么值
  - When：什么时候才能将已写入的值返回给读操作
- 需要满足以下条件
  - 处理器P对单元X进行一次写之后又对单元X进行读，读和写之间没有其它处理器对单元X进行写，则P读到的值总是前面写进去的值。

- 处理器P对单元X进行写之后，另一处理器Q对单元X进行读，读和写之间无其它写，则Q读到的值应为P写进去的值。
- 对同一单元的写是串行化的，即任意两个处理器对同一单元的两次写，从各个处理器的角度看来顺序都是相同的。（写串行化 ）

➢ 在后面的讨论中，我们假设：

- 直到所有的处理器均看到了写的结果，这个写操作才算完成；
- 处理器的任何访存均不能改变写的顺序。就是说，允许处理器对读进行重排序，但必须以程序规定的顺序进行写。

10.2.2 实现一致性的基本方案

在一致的多处理机中，Cache提供两种功能：

- 共享数据的迁移

  减少了对远程共享数据的访问延迟，也减少了对共享存储器带宽的要求。

- 共享数据的复制

  不仅减少了访问共享数据的延迟，也减少了访问共享数据所产生的冲突。

一般情况下，小规模多处理机是采用硬件的方法来实现Cache的一致性。

1. Cache一致性协议

在多个处理器中用来维护一致性的协议。

- 关键：跟踪记录共享数据块的状态
- 两类协议（采用不同的技术跟踪共享数据的状态）
  - 目录式协议（directory）

    物理存储器中数据块的共享状态被保存在一个称为目录的地方。
  - 监听式协议（snooping）
    - 每个Cache除了包含物理存储器中块的数据拷贝之外，也保存着各个块的共享状态信息。

- Cache通常连在共享存储器的总线上，当某个Cache需要访问存储器时，它会把请求放到总线上广播出去，其他各个Cache控制器通过监听总线（它们一直在监听）来判断它们是否有总线上请求的数据块。如果有，就进行相应的操作。

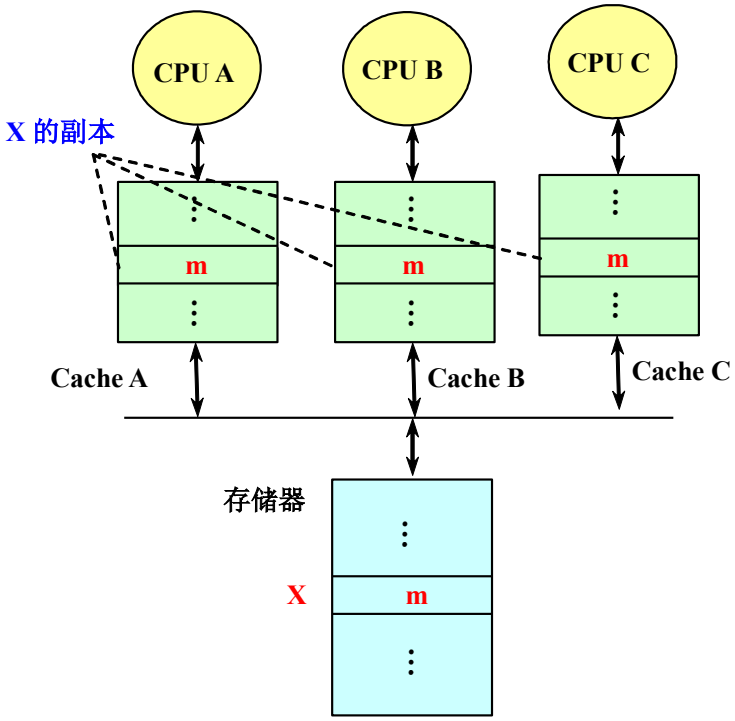
2. 采用两种方法来解决Cache一致性问题。

➢ 写作废协议

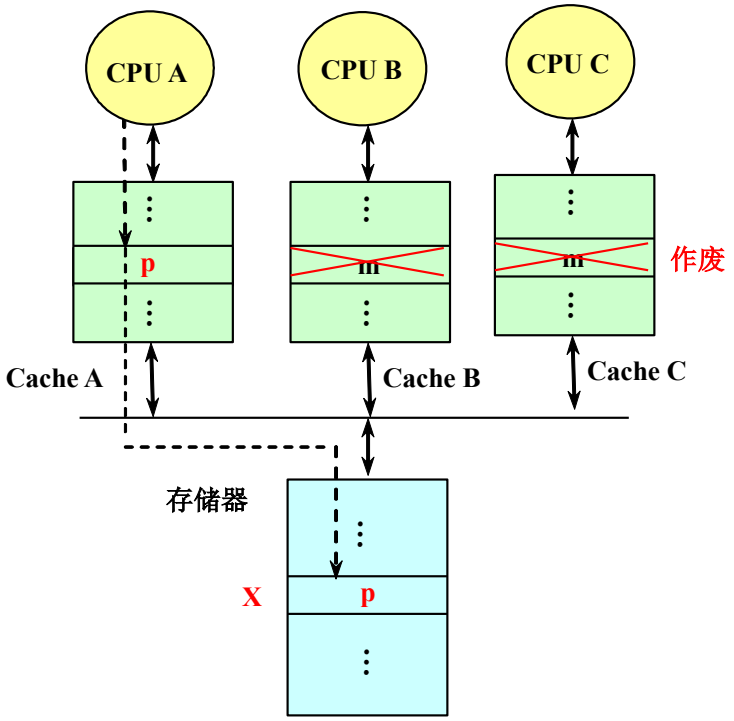
在处理器对某个数据项进行写入之前，保证它拥有对该数据项的唯一的访问权。(作废其它的副本)

例 监听总线、写作废协议举例（采用写直达法）

初始状态：CPU A、CPU B、CPU C都有X的副本。在CPU A要对X进行写入时，需先作废CPU B和CPU C中的副本，然后再将p写入Cache A中的副本，同时用该数据更新主存单元X。



（a） CPU A 写入前



（b） CPU A 将 p 写入 X 后，作废其他 Cache 中的副本

➢ 写更新协议

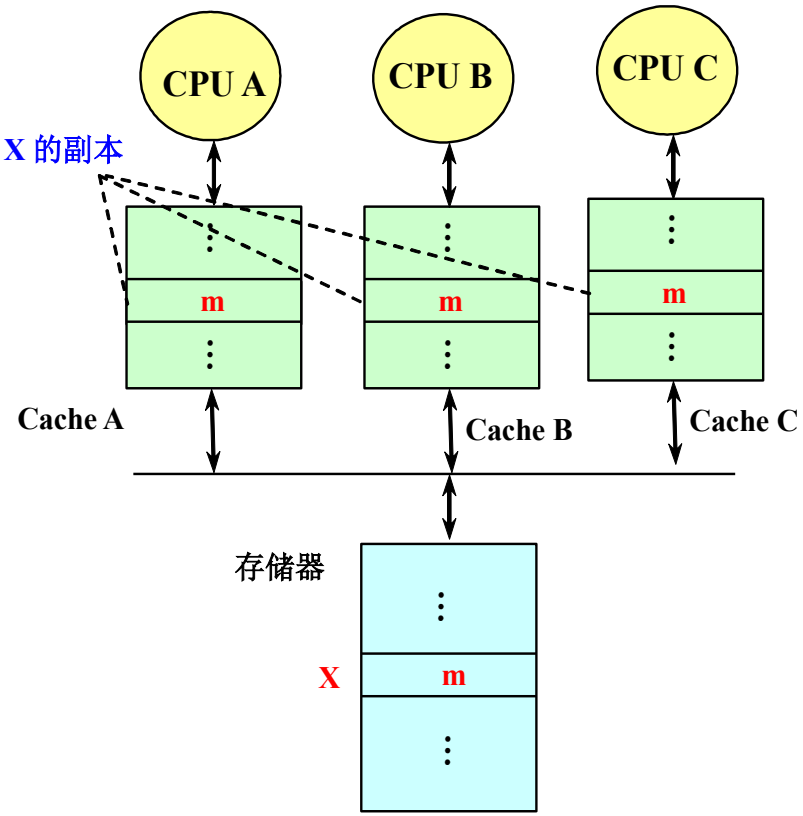
当一个处理器对某数据项进行写入时，通过广播使其它Cache中所有对应于该数据项的副本进行更新。

例 监听总线、写更新协议举例（采用写直达法）

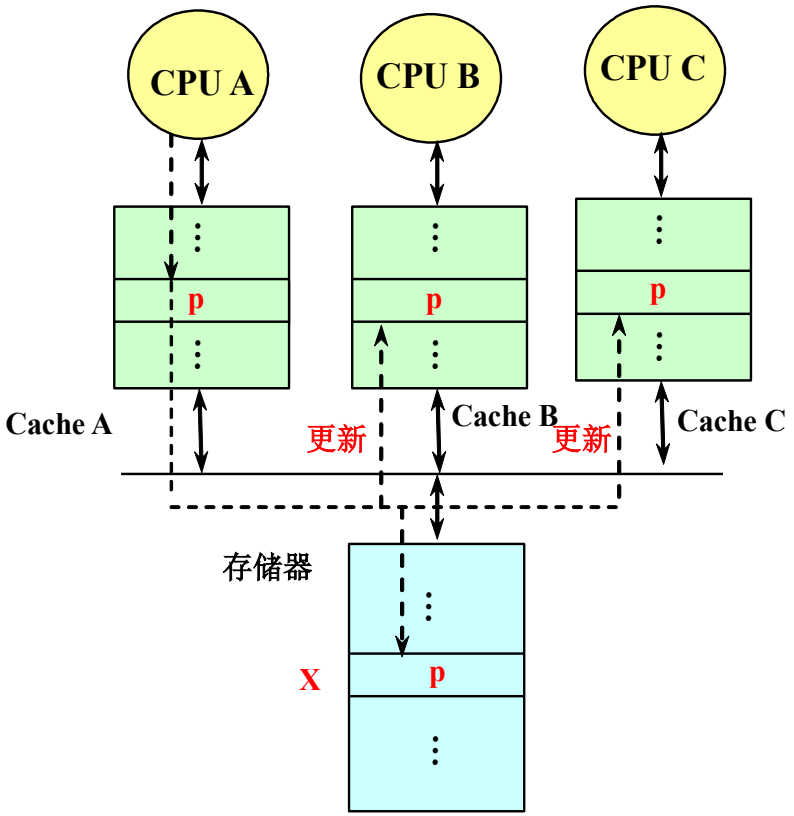
假设：3个Cache都有X的副本。

当CPU A将数据p写入Cache A中的副本时，将p广播给所有的Cache，这些Cache用p更新其中的副本。

由于这里是采用写直达法，所以CPU A还要将p写入存储器中的X。如果采用写回法，则不需要写入存储器。



（a） CPU A 写入前



（b） CPU A 将 p 写入 X 后，更新其他 Cache 中的副本

- 写更新和写作废协议性能上的差别主要来自：
  - 在对同一个数据进行多次写操作而中间无读操作的情况下，写更新协议需进行多次写广播操作，而写作废协议只需一次作废操作。
  - 在对同一Cache块的多个字进行写操作的情况下，写更新协议对于每一个写操作都要进行一次广播，而写作废协议仅在对该块的第一次写时进行作废操作即可。

    写作废是针对Cache块进行操作，而写更新则是针对字（或字节）进行。
  - 考虑从一个处理器A进行写操作后到另一个处理器B能读到该写入数据之间的延迟时间。

    写更新协议的延迟时间较小。

10.2.3 监听协议的实现

1. 监听协议的基本实现技术

- 实现监听协议的关键有3个方面
  - 处理器之间通过一个可以实现广播的互连机制相连。

    通常采用的是总线。
  - 当一个处理器的Cache响应本地CPU的访问时，如果它涉及到全局操作，其Cache控制器就要在获得总线的控制权后，在总线上发出相应的消息。
  - 所有处理器都一直在监听总线，它们检测总线上的地址在它们的Cache中是否有副本。若有，则响应该消息，并进行相应的操作 。

- 写操作的串行化：由总线实现

（获取总线控制权的顺序性）

2. Cache发送到总线上的消息主要有以下两种：
   - **RdMiss**——读不命中
   - **WtMiss**——写不命中
- 需要通过总线找到相应数据块的最新副本，然后调入本地Cache中。
  - 写直达Cache：因为所有写入的数据都同时被写回主存，所以从主存中总可以取到其最新值。
  - 对于写回Cache，得到数据的最新值会困难一些，因为最新值可能在某个Cache中，也可能在主存中。

（后面的讨论中，只考虑写回法Cache）

- 有的监听协议还增设了一条Invalidate消息，用来通知其他各处理器作废其Cache中相应的副本。
  - 与WtMiss的区别：Invalidate不引起调块
- Cache的标识（tag）可直接用来实现监听。
- 作废一个块只需将其有效位置为无效。
- 给每个Cache块增设一个共享位
  - 为“1”：该块是被多个处理器所共享
  - 为“0”：仅被某个处理器所独占

块的拥有者：拥有该数据块的唯一副本的处理器。

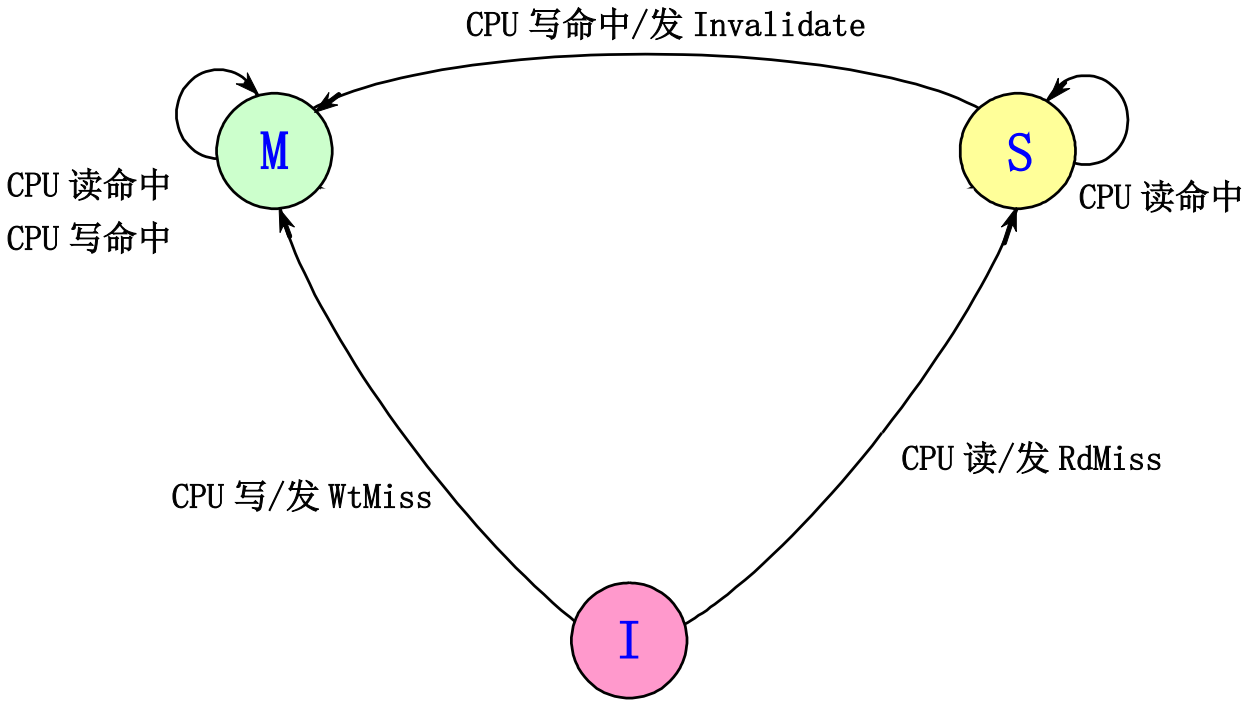
3. 监听协议举例

- 在每个结点内嵌入一个有限状态控制器。
  - 该控制器根据来自处理器或总线的请求以及Cache块的状态，做出相应的响应。
- 每个数据块的状态取以下3种状态中的一种：
  - 无效（简称I）：Cache中该块的内容为无效。
  - 共享（简称S）：该块可能处于共享状态。
    - 在多个处理器中都有副本。这些副本都相同，且与存储器中相应的块相同。
  - 已修改（简称M）：该块已经被修改过，并且还没写入存储器。

    （块中的内容是最新的，系统中唯一的最新副本）

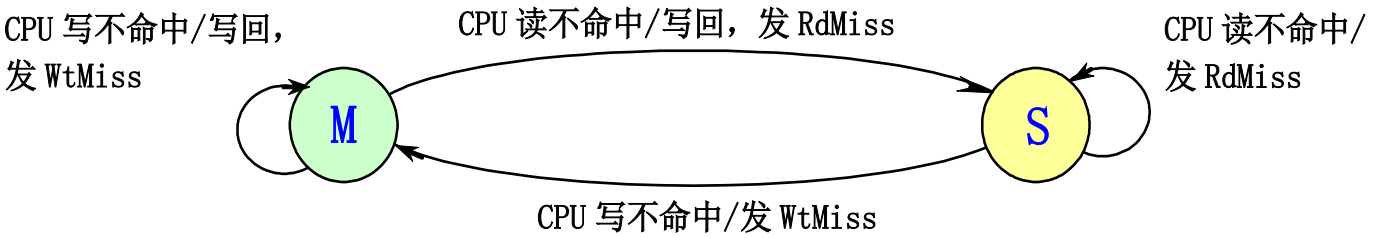
下面来讨论在各种情况下监听协议所进行的操作。

- 响应来自处理器的请求
  - 不发生替换的情况



写作废协议中（采用写回法），Cache块的状态转换图

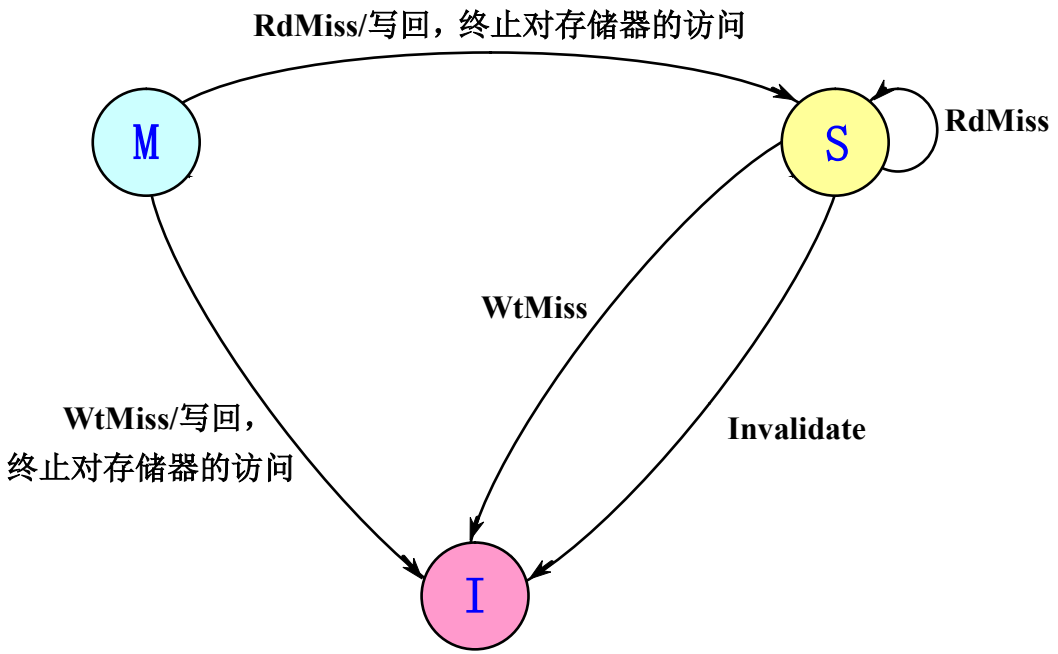
- 发生替换的情况



写作废协议中（采用写回法），Cache块的状态转换图

➢ 响应来自总线的请求

- 每个处理器都在监视总线上的消息和地址，当发现有与总线上的地址相匹配的Cache块时，就要根据该块的状态以及总线上的消息，进行相应的处理。



写作废协议中（采用写回法），Cache块的状态转换图

10.3 分布式共享存储器系统结构

10.3.1 目录协议的基本思想

- 广播和监听的机制使得监听一致性协议的可扩放性很差。
- 寻找替代监听协议的一致性协议。

（采用目录协议）

1. 目录协议

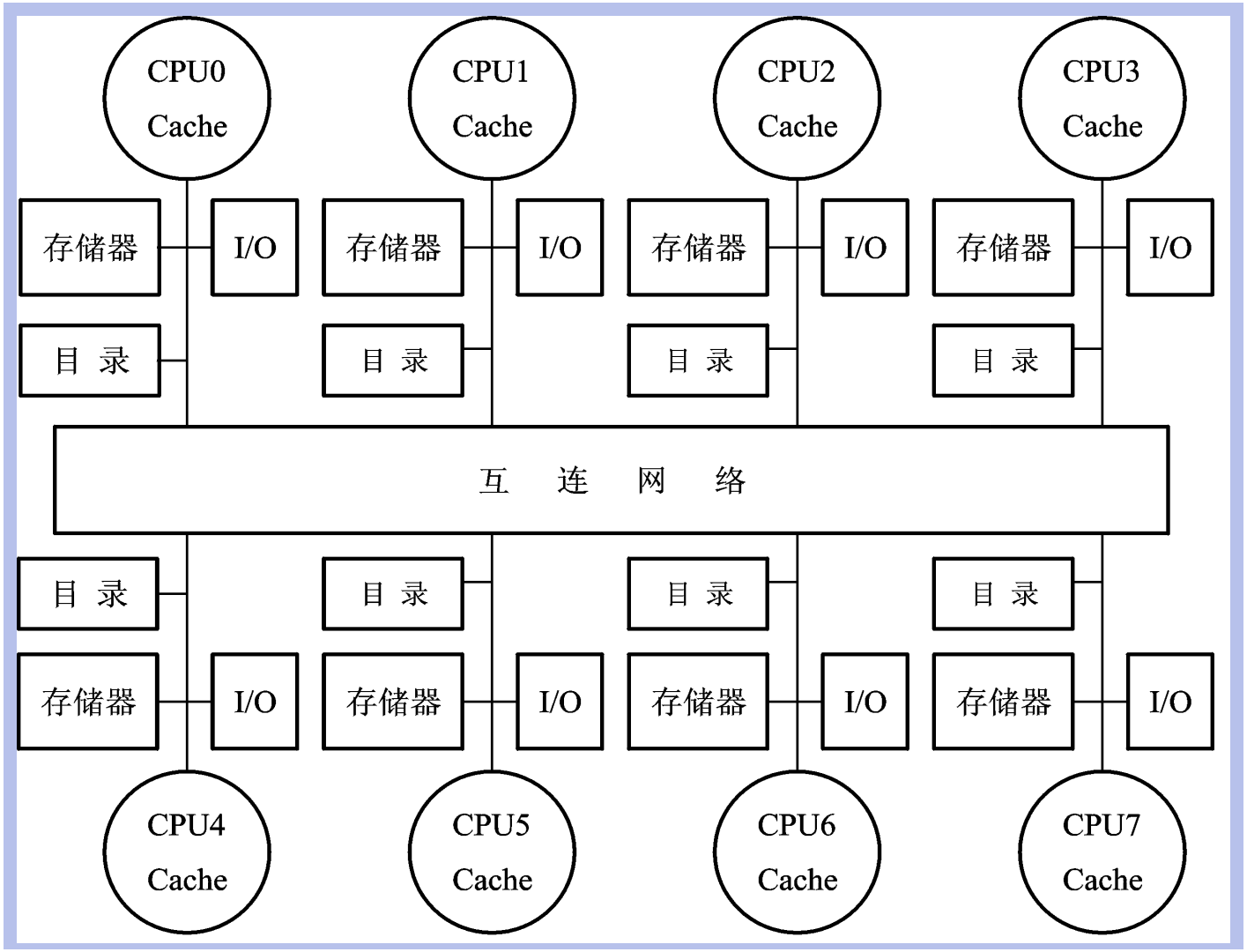
- 目录：一种集中的数据结构。对于存储器中的每一个可以调入Cache的数据块，在目录中设置一条目录项，用于记录该块的状态以及哪些Cache中有副本等相关信息。

- 特点：

  对于任何一个数据块，都可以快速地在唯一的一个位置中找到相关的信息。这使一致性协议避免了广播操作。

- 位向量：记录哪些Cache中有副本。
  - 每一位对应于一个处理器。
  - 长度与处理器的个数成正比。
  - 由位向量指定的处理机的集合称为共享集S。
- 分布式目录
  - 目录与存储器一起分布到各结点中，从而对于不同目录内容的访问可以在不同的结点进行。

- 对每个结点增加目录后的分布式存储器多处理机



- 目录法最简单的实现方案：对于存储器中每一块都在目录中设置一项。目录中的信息量与M×N成正比。

  其中：

  - M：存储器中存储块的总数量
  - N：处理器的个数
  - 由于M=K×N， K是每个处理机中存储块的数量，所以如果K保持不变，则目录中的信息量就与N^2成正比。

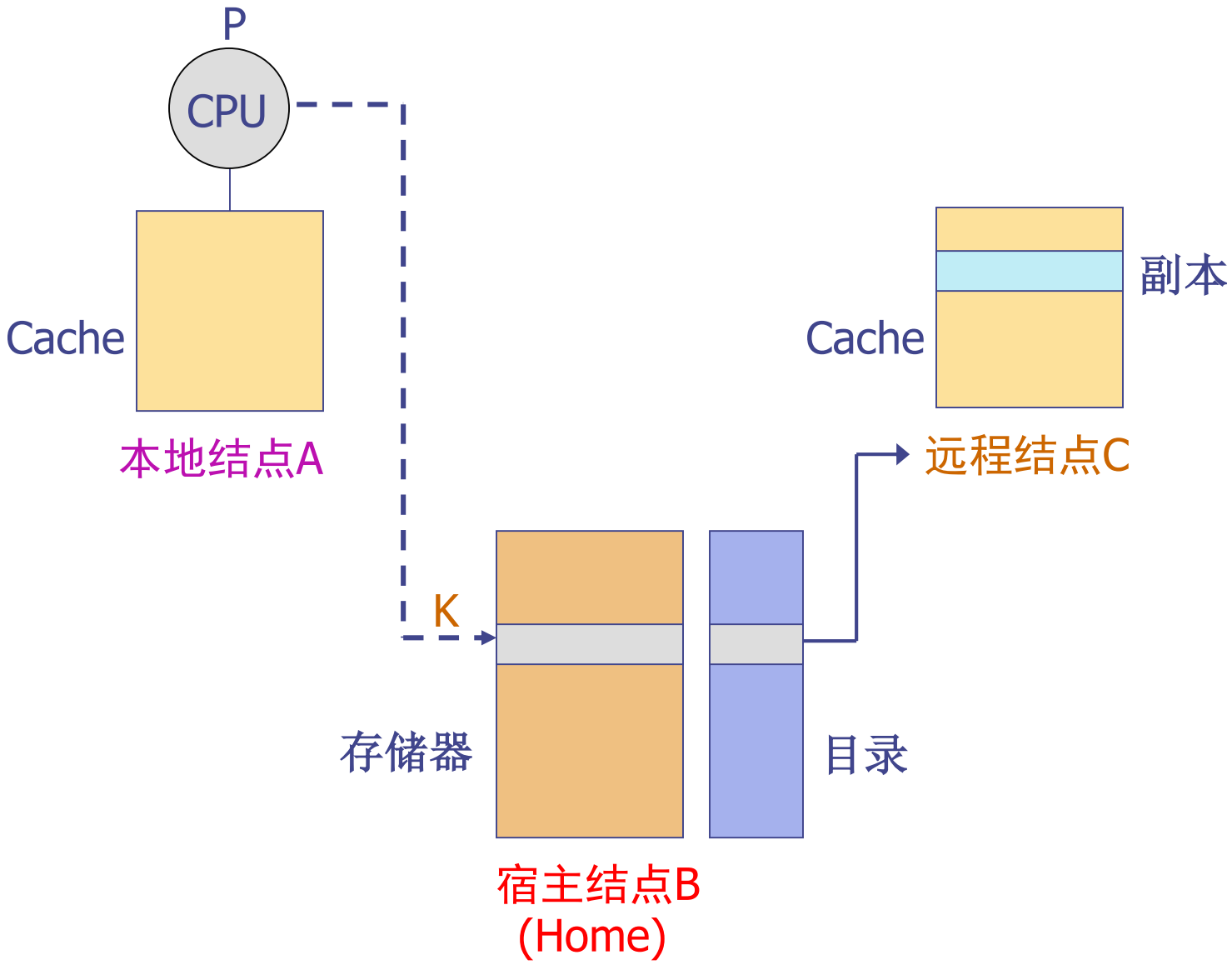
2. 在目录协议中，存储块的状态有3种：

- 未缓冲：该块尚未被调入Cache。所有处理器的Cache中都没有这个块的副本。
- 共享：该块在一个或多个处理机上有这个块的副本，且这些副本与存储器中的该块相同。
- 独占：仅有一个处理机有这个块的副本，且该处理机已经对其进行了写操作，所以其内容是最新的，而存储器中该块的数据已过时。

这个处理机称为该块的拥有者。

3. 本地结点、宿主结点以及远程结点的关系

- 本地结点：发出访问请求的结点
- 宿主结点：包含所访问的存储单元及其目录项的结点
- 远程结点可以和宿主结点是同一个结点，也可以不是同一个结点。



宿主结点：存放有对应地址的存储器块和目录项的结点

4. 在结点之间发送的消息

- 本地结点发给宿主结点（目录）的消息

说明：括号中的内容表示所带参数。

P：发出请求的处理机编号

K：所要访问的地址

- **RdMiss（P，K）**

处理机P读取地址为A的数据时不命中，请求宿主结点提供数据（块），并要求把P加入共享集。

- **WtMiss（P，K）**

处理机P对地址A进行写入时不命中，请求宿主结点提供数据，并使P成为所访问数据块的独占者。

- **Invalidate（K）**

  请求向所有拥有相应数据块副本（包含地址*K*）的远程Cache发Invalidate消息，作废这些副本。

➢ 宿主结点（目录）发送给远程结点的消息

- **Invalidate（K）**

  作废远程Cache中包含地址K的数据块。

- **Fetch（K）**

  从远程Cache中取出包含地址*K*的数据块，并将之送到宿主结点。把远程Cache中那个块的状态改为“共享”。

- **Fetch&Inv（K）**

- 从远程Cache中取出包含地址*K*的数据块，并将之送到宿主结点。然后作废远程Cache中的那个块。

➢ 宿主结点发送给本地结点的消息

DReply（D）

- D表示数据内容。
- 把从宿主存储器获得的数据返回给本地Cache。

➢ 远程结点发送给宿主结点的消息

WtBack（K，D）

- 把远程Cache中包含地址*K*的数据块写回到宿主结点中， 该消息是远程结点对宿主结点发来的“取数据”或“取/作废”消息的响应。

- 本地结点发送给被替换块的宿主结点的消息
  - **MdSharer（P，K）**

    用于当本地Cache中需要替换一个包含地址K的块、且该块未被修改过的情况。这个消息发给该块的宿主结点，请求它将P从共享集中删除。如果删除后共享集变为空集，则宿主结点还要将该块的状态改变为“未缓存”（U）。
  - **WtBack2（P，K，D）**

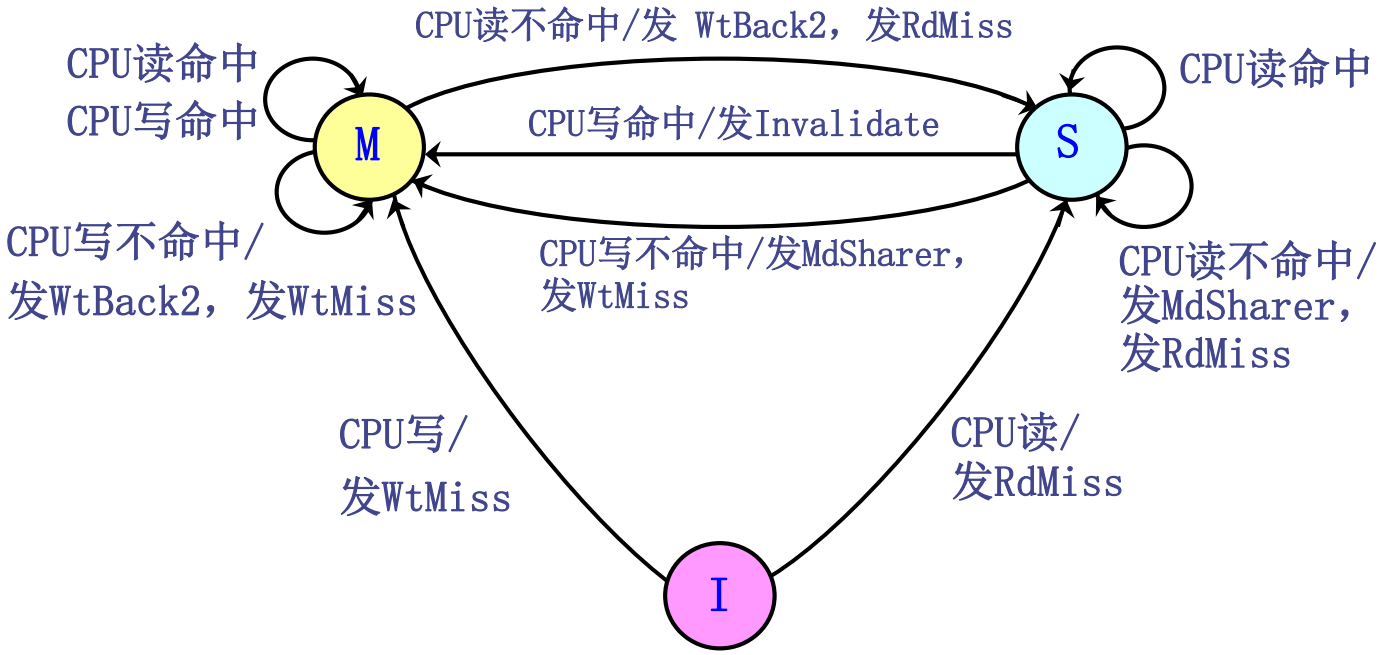
    用于当本地Cache中需要替换一个包含地址K的块、且该块已被修改过的情况。这个消息发给该块的宿主结点，完成两步操作：①把该块写回；②进行与MdSharer相同的操作。

10.3.2 目录协议实例

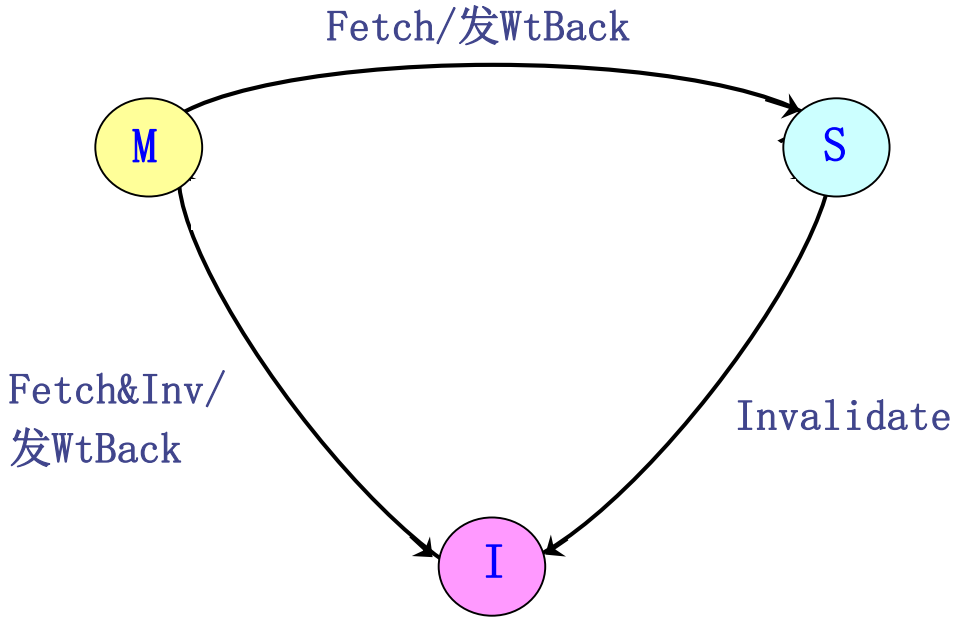
- 在基于目录的协议中，目录承担了一致性协议操作的主要功能。
  - 本地结点把请求发给宿主结点中的目录，再由目录控制器有选择地向远程结点发出相应的消息。
  - 发出的消息会产生两种不同类型的动作：
    - 更新目录状态
    - 使远程结点完成相应的操作

1. 在基于目录协议的系统中， Cache块的状态转换图。

   - 响应本地Cache CPU请求



- 远程结点中Cache块响应来自宿主结点的请求的状态转换图



2. 目录的状态转换及相应的操作

如前所述：

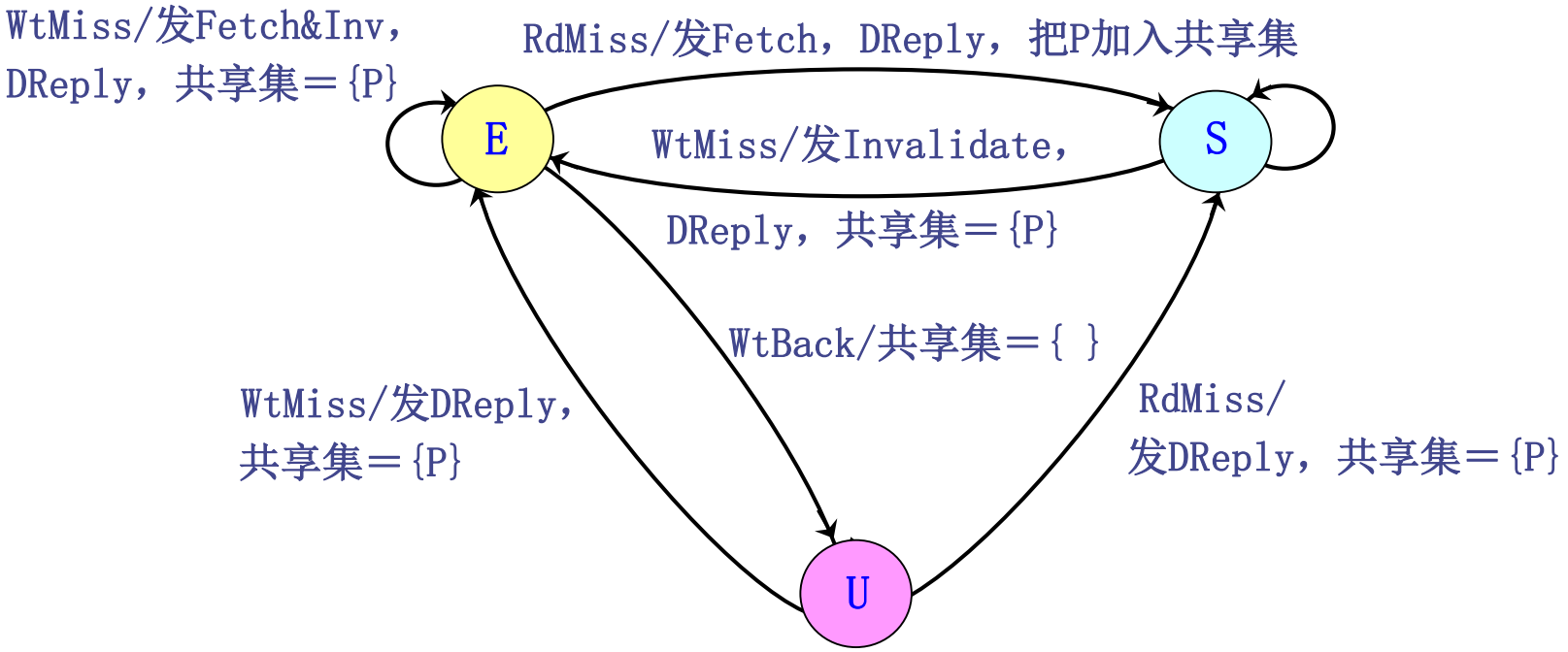
- 目录中存储器块的状态有3种
  - 未缓存
  - 共享
  - 独占
- 位向量记录拥有其副本的处理器的集合。这个集合称为共享集合。
- 对于从本地结点发来的请求，目录所进行的操作包括：

- 向远程结点发送消息以完成相应的操作。这些远程结点由共享集合指出；
- 修改目录中该块的状态；
- 更新共享集合。

➢ 目录可能接收到3种不同的请求

- 读不命中
- 写不命中
- 数据写回

（假设这些操作是原子的）



U：未缓存（Uncached）　　S：共享（Shared）：只读

E：独占（Exclusive）：可读写　　P：本地处理器

目录的状态转换及相应的操作

- 当一个块处于未缓存状态时，对该块发出的请求及处理操作为：
  - RdMiss（读不命中）
    - 将所要访问的存储器数据送往请求方处理机，且该处理机成为该块的唯一共享结点，本块的状态变成共享。
  - WtMiss（写不命中）
    - 将所要访问的存储器数据送往请求方处理机，该块的状态变成独占，表示该块仅存在唯一的副本。其共享集合仅包含该处理机，指出该处理机是其拥有者。

- 当一个块处于共享状态时，其在存储器中的数据是当前最新的，对该块发出的请求及其处理操作为：
  - **RdMiss**
    - 将存储器数据送往请求方处理机，并将其加入共享集合。
  - **WtMiss**
    - 将数据送往请求方处理机，对共享集合中所有的处理机发送作废消息，且将共享集合改为仅含有该处理机，该块的状态变为独占。

- 当某块处于独占状态时，该块的最新值保存在共享集合所指出的唯一处理机（拥有者）中。

  有三种可能的请求：

  - **RdMiss**
    - 将“取数据”的消息发往拥有者处理机，将它所返回给宿主结点的数据写入存储器，并进而把该数据送回请求方处理机，将请求方处理机加入共享集合。
    - 此时共享集合中仍保留原拥有者处理机（因为它仍有一个可读的副本）。
    - 将该块的状态变为共享。

- **WtMiss**
  - 该块将有一个新的拥有者。
  - 给旧的拥有者处理机发送消息，要求它将数据块送回宿主结点写入存储器，然后再从该结点送给请求方处理机。
  - 同时还要把旧拥有者处理机中的该块作废。把请求处理机加入共享者集合，使之成为新的拥有者。
  - 该块的状态仍旧是独占。
- **WtBack**（写回）
  - 当一个块的拥有者处理机要从其Cache中把该块替换出去时，必须将该块写回其宿主结点的

存储器中，从而使存储器中相应的块中存放的数据是最新的（宿主结点实际上成为拥有者）；

- 该块的状态变成未缓冲，其共享集合为空。

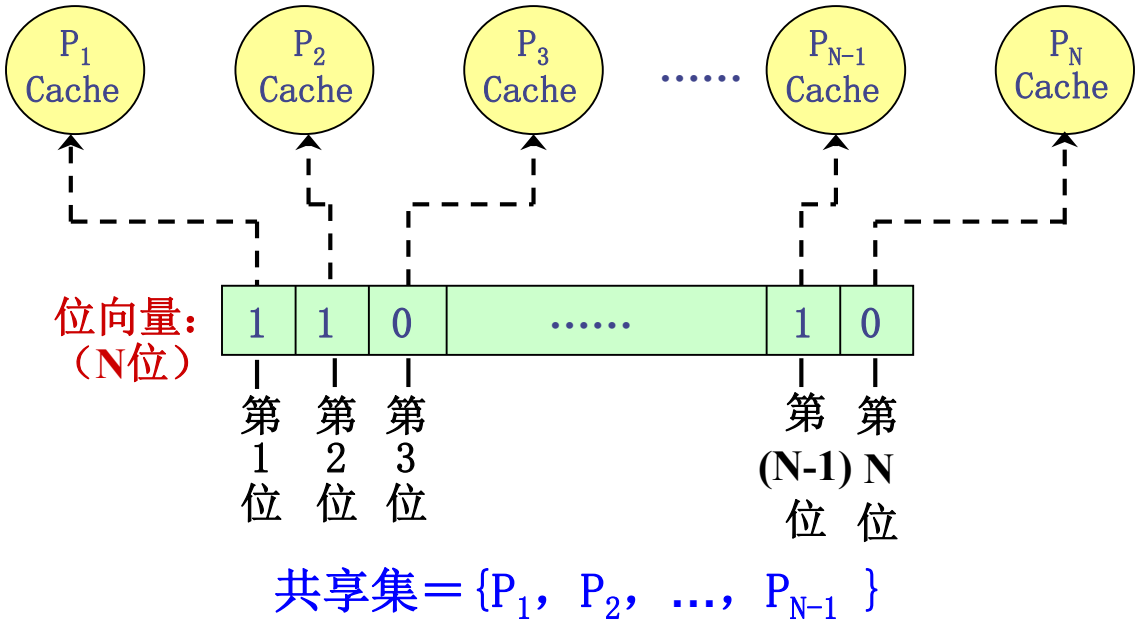
10.3.3 目录的三种结构

- 不同目录协议的主要区别主要有两个
  - 所设置的存储器块的状态及其个数不同
  - 目录的结构
- 目录协议分为3类

  全映象目录、有限映象目录、链式目录

1. 全映象目录

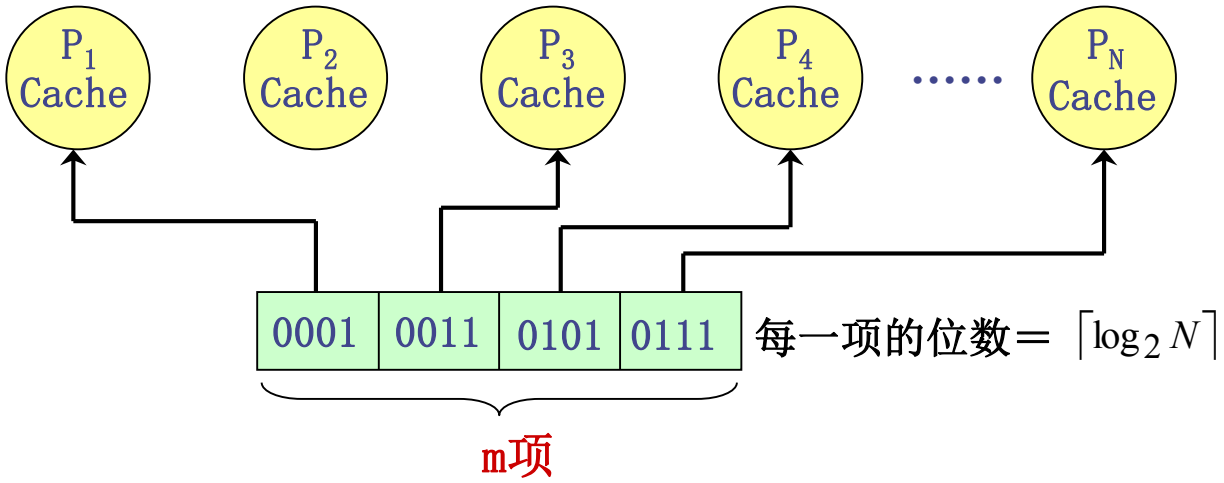
- 每一个目录项都包含一个N位（N为处理机的个数）的位向量，其每一位对应于一个处理机。
  - 举例
- 优点：处理比较简单，速度也比较快。
- 缺点：
  - 存储空间的开销很大。
  - 目录项的数目与处理机的个数N成正比，而目录项的大小（位数）也与N成正比，因此目录所占用的空间与N^2成正比。
  - 可扩放性很差。



- 当位向量中的值为“1”时，就表示它所对应的处理机有该数据块的副本；否则就表示没有。
- 在这种情况下，共享集合由位向量中值为“1”的位所对应的处理机构成。

2. 有限映象目录

- 提高其可扩放性和减少目录所占用的空间。
- 核心思想：采用位数固定的目录项目
  - 限制同一数据块在所有Cache中的副本总数。
  - 例如，限定为常数m。则目录项中用于表示共享集合所需的二进制位数为：$m \times \log_2 N$。
  - 目录所占用的空间与$N \times \lceil \log_2 N \rceil$成正比。
- 举例



共享集＝{P_1，P_3，P_4，P_7 }

有限映象目录（m＝4，N≥8的情况）

- 缺点
  - 当同一数据的副本个数大于m时，必须做特殊处理。当目录项中的m个指针都已经全被占满，而某处理机又需要新调入该块时，就需要在其m个指针中选择一个，将之驱逐，以便腾出位置，存放指向新调入块的处理机的指针 。

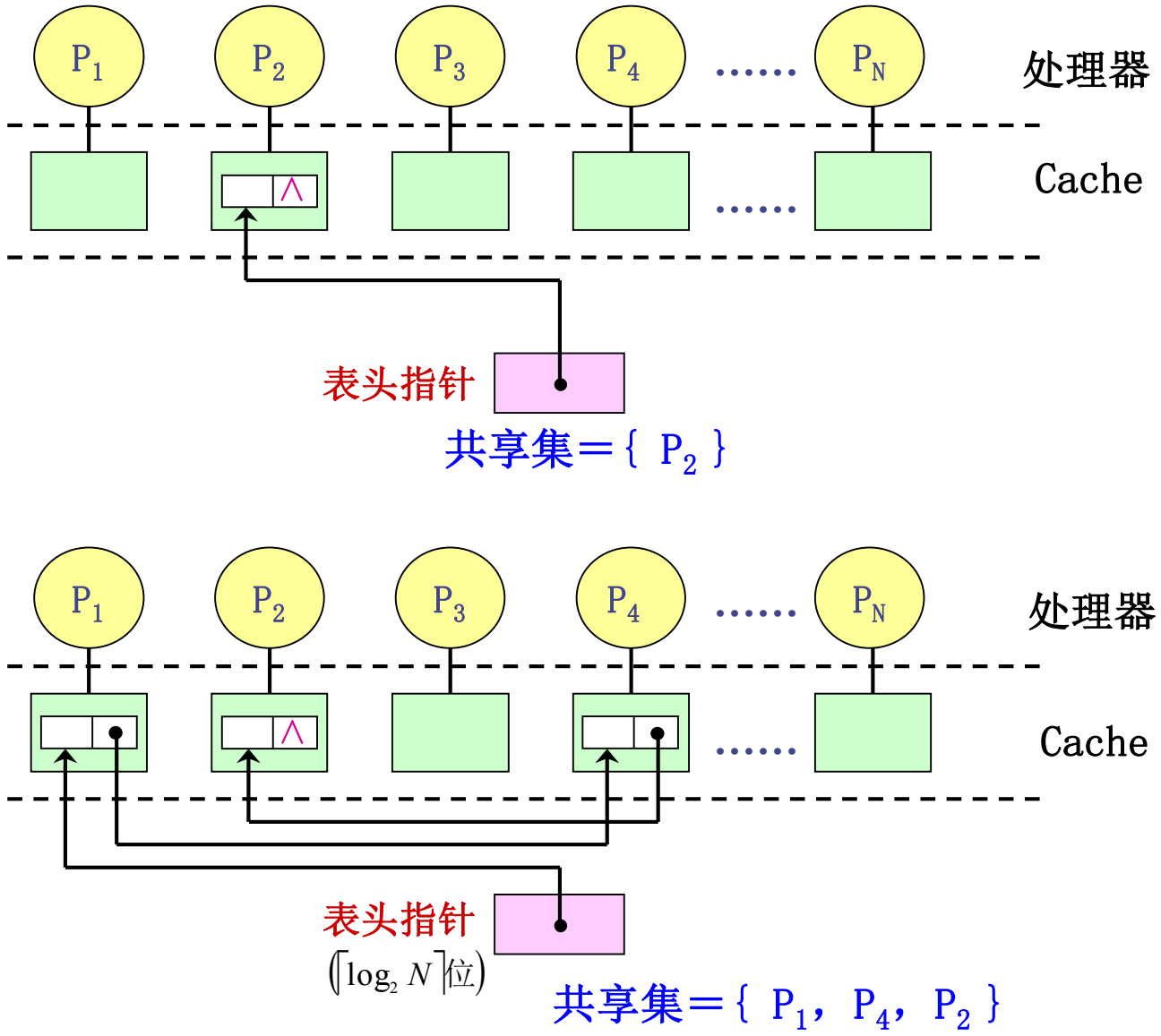
3. 链式目录

- 用一个目录指针链表来表示共享集合。当一个数据块的副本数增加（或减少）时，其指针链表就跟着变长（或变短）。
- 由于链表的长度不受限制，因而带来了以下优点：既不限制副本的个数，又保持了可扩展性。

- 链式目录有两种实现方法
  - 单链法

    当Cache中的块被替换出去时，需要对相应的链表进行操作——把相应的链表元素（假设是链表中的第i个）删除。实现方法有以下两种：
    - 沿着链表往下寻找第i个元素，找到后，修改其前后的链接指针，跳过该元素。
    - 找到第i个元素后，作废它及其后的所有元素所对应的Cache副本。
  - 双链法
    - 在替换时不需要遍历整个链表。
    - 节省了处理时间，但其指针增加了一倍，而且一致性协议也更复杂了。

采用单向链法的示意图


P1
P2
P3
P4
……
PN
处理器
Cache
表头指针
共享集＝{ P2 }
P1
P2
P3
P4
……
PN
处理器
Cache
表头指针
(⌈log2 N⌉位)
共享集＝{ P1，P4，P2 }

10.4 同 步

同步机制通常是在硬件提供的同步指令的基础上，通过用户级软件例程来建立的。

10.4.1 基本硬件原语

在多处理机中实现同步，所需的主要功能是：

- 一组能以原子操作的方式读出并修改存储单元的硬件原语。它们都能以原子操作的方式读/修改存储单元，并指出所进行的操作是否以原子的方式进行。
- 通常情况下，用户不直接使用基本的硬件原语，原语主要供系统程序员用来编制同步库函数。

1. 典型操作：原子交换（atomic exchange）

- 功能：将一个存储单元的值和一个寄存器的值进行交换。

  建立一个锁，锁值：

  - 0：表示开的（可用）
  - 1：表示已上锁（不可用）

- 处理器上锁时，将对应于该锁的存储单元的值与存放在某个寄存器中的1进行交换。如果返回值为0，存储单元的值此时已置换为1，防止了别的进程竞争该锁。
- 实现同步的关键：操作的原子性

2. 测试并置定（test_and_set）
   - 先测试一个存储单元的值，如果符合条件则修改其值。
3. 读取并加1（fetch_and_increment）
   - 它返回存储单元的值并自动增加该值。
4. 使用指令对
   - LL(load linked或load locked)的取指令
   - SC(store conditional)的特殊存指令

- 指令顺序执行：
  - 如果由LL指明的存储单元的内容在SC对其进行写之前已被其它指令改写过，则第二条指令SC执行失败；
  - 如果在两条指令间进行切换也会导致SC执行失败。
  - SC将返回一个值来指出该指令操作是否成功：
    - “1”：成功
    - “0”：不成功
  - LL则返回该存储单元初始值。

例：实现对由R1指出的存储单元进行原子交换操作。

```
try: OR    R3, R4, R0      // R4中为交换值。把该值送入R3
     LL    R2, 0（R1）
                           // 把单元0（R1）中的值取到R2
     SC    R3, 0（R1）     // 若0（R1）中的值与R3中的值相
                              同，则置R3的值为1，否则置为0
     BEQZ  R3, try         // 存失败（R3的值为0）则转移
     MOV   R4, R2          // 将取的值送往R4
```

最终R4和由R1指向的单元值进行原子交换，在LL和SC之间如有别的处理器插入并修改了存储单元的值，SC将返回0并存入R3中，从而使这段程序再次执行。

- LL/SC机制的一个优点：用来构造别的同步原语

例如：构造原子操作fetch_and_increment：

```
try:    LL      R2, 0(R1)
        DADDIU  R2, R2, #1
        SC      R2, 0(R1)
        BEQZ    R2, try
```

- 指令对的实现必须跟踪地址

由LL指令指定一个寄存器，该寄存器存放着一个单元地址，这个寄存器常称为连接寄存器。

10.4.2 用一致性实现锁

- 采用多处理机的一致性机制来实现旋转锁。
- 旋转锁

  处理器环绕一个锁不停地旋转而请求获得该锁。适合于这样的场合：锁被占用的时间很少，在获得锁后加锁过程延迟很小。

1. 无Cache一致性机制

在存储器中保存锁变量，处理器可以不断地通过一个原子操作请求使用权。

比如：利用原子交换操作，并通过测试返回值而知道锁的使用情况。释放锁的时候，处理器只需简单地将锁置为0。

例：用原子交换操作对旋转锁进行加锁，R1中存放的是该旋转锁的地址。

```
        DADDIU    R2, R0, #1
lockit: EXCH      R2, 0(R1)
        BNEZ      R2, lockit
```

2. 机器支持Cache一致性

- 将锁调入Cache，并通过一致性机制使锁值保持一致。
- 优点：
  - 可使“环绕”的进程只对本地Cache中的锁（副本）进行操作，而不用在每次请求占用锁时都进行一次全局的存储器访问；
  - 可利用访问锁时所具有的局部性，即处理器最近使用过的锁不久又会使用。

    （减少为获得锁而花费的时间）

- 改进旋转锁（获得第一条好处）
  - 只对本地Cache中锁的副本进行读取和检测，直到发现该锁已经被释放。然后，该程序立即进行交换操作，去跟在其它处理器上的进程争用该锁变量。
  - 修改后的旋转锁程序：

```
lockit:  LD      R2, 0(R1)
         BNEZ    R2, lockit
         DADDIU  R2, R0, #1
         EXCH    R2, 0(R1)
         BNEZ    R2, lockit
```

  - 3个处理器利用原子交换争用旋转锁所进行的操作

3个处理器利用原子交换争用旋转锁所进行的操作

步骤	处理器P0	处理器P1	处理器P2	锁的状态	总线/目录操作
1	占有锁	环绕测试 是否lock=0	环绕测试 是否lock=0	共享	无
2	将锁置为0	（收到作废命令）	（收到作废命令）	专有（P0）	P0发出对锁变量的作废消息
3		Cache不命中	Cache不命中	共享	总线/目录收到P2 Cache不命中；锁从P0写回
4		（因总线/目录忙而等待）	lock=0	共享	P2 Cache不命中被处理
5		Lock=0	执行交换， 导致Cache不命中	共享	P1 Cache不命中被处理
6		执行交换， 导致Cache不命中	交换完毕：返回0 并置lock=1	专有（P2）	总线/目录收到P2 Cache不命中；发作废消息
7		交换完毕： 返回1	进入关键程序段	专有（P1）	总线/目录处理P1 Cache不命中；写回
8		环绕测试 是否lock=0			无

- LL / SC原语的另一个优点：读写操作明显分开

  LL不产生总线数据传送，这使下面代码与使用经过优化交换的代码具有相同的特点：

```
lockit:   LL      R2, 0(R1)
          BNEZ    R2, lockit
          DADDIU  R2, R0, #1
          SC      R2, 0(R1)
          BEQZ    R2, lockit
```

第一个分支形成环绕的循环体，第二个分支解决了两个处理器同时看到锁可用的情况下的争用问题。尽管旋转锁机制简单并且具有吸引力，但难以将它应用于处理器数量很多的情况。

10.4.3 同步性能问题

简单旋转锁不能很好地适应可缩扩性。大规模多处理机中，若所有的处理器都同时争用同一个锁，则会导致大量的争用和通信开销。

例10.3 假设某条总线上有10个处理器同时准备对同一变量加锁。如果每个总线事务处理（读不命中或写不命中）的时间是100个时钟周期，而且忽略对已调入Cache中的锁进行读写的时间以及占用该锁的时间。

（1）假设该锁在时间为0时被释放，并且所有处理器都在旋转等待该锁。问：所有10个处理器都获得该锁所需的总线事务数目是多少？

（2）假设总线是非常公平的，在处理新请求之前，要先全部处理好已有的请求。并且各处理器的速度相同。问：处理10个请求大概需要多少时间？

解 当i个处理器争用锁的时候，它们都各自完成以下操作序列，每一个操作产生一个总线事务：

- 访问该锁的i个LL指令操作
- 试图占用该锁（并上锁）的i个SC指令操作
- 1个释放锁的存操作指令

因此对于i个处理器来说，一个处理器获得该锁所要进行的总线事务的个数为2i+1。

由此可知，对n个处理器，总的总线事务个数为：

$$\sum_{i=1}^{n}(2i+1)=n(n+1)+n=n^2+2n$$

对于10个处理器来说，其总线事务数为120个，需要12000个时钟周期。

- 本例中问题的根源：锁的争用、对锁进行访问的串行性以及总线访问的延迟。
- 旋转锁的主要优点：总线开销或网络开销比较低，而且当一个锁被同一个处理器重用时具有很好的性能。

1. 如何用旋转锁来实现一个常用的高级同步原语：栅栏
   - 栅栏强制所有到达该栅栏的进程进行等待，直到全部的进程到达栅栏，然后释放全部的进程，从而形成同步。

- 栅栏的典型实现

  用两个旋转锁：

  - 用来保护一个计数器，它记录已到达该栅栏的进程数；
  - 用来封锁进程直至最后一个进程到达该栅栏。

- 一种典型的实现

  其中：

  - lock和unlock提供基本的旋转锁
  - 变量count记录已到达栅栏的进程数
  - total规定了要到达栅栏的进程总数

```
lock (counterlock) ;                    //  确保更新的原子性
    if (count==0) release=0;    //  第一个进程则重置release
    count=count+1;                      //  到达进程数加1
    unlock (counterlock) ;              //  释放锁
    if (count==total) {                 //  进程全部到达
        count=0;                        //  重置计数器
        release=1;                      //  释放进程
        }
    else {                              //  还有进程未到达
        spin (release=1) ;              //  等待别的进程到达
        }
```

- 对counterlock加锁保证增量操作的原子性。
- release用来封锁进程直到最后一个进程到达栅栏。
- spin（release=1）使进程等待直到全部的进程到达栅栏。

➢ 实际情况中会出现的问题

栅栏通常是在循环中使用，从栅栏释放的进程运行一段后又会再次返回栅栏，这样有可能出现某个进程永远离不开栅栏的状况(它停在旋转操作上)。

- 一种解决方法

  当进程离开栅栏时进行计数（和到达时一样），在上次栅栏使用中的所有进程离开之前，不允许任何进程重用并初始化本栅栏。但这会明显增加栅栏的延迟和竞争。

- 另一种解决办法
  - 采用sense_reversing栅栏，每个进程均使用一个私有变量local_sense，该变量初始化为1。
  - sense_reversing栅栏的代码

优缺点：使用安全，但性能比较差。

对于10个处理器来说，当同时进行栅栏操作时，如果忽略对Cache的访问时间以及其它非同步操作所需的时间，则其总线事务数为204个，如果每个总线事物需要100个时钟周期，则总共需要20400个时钟周期。

```
local_sense=! local_sense;              //  local-sense取反
    lock (counterlock) ;                //  确保更新的原子性
    count++;                            //  到达进程数加1
    unlock (counterlock) ;              //  释放锁
    if (count==total) {                 //  进程全部到达
        count=0;                        //  重置计数器
        release=local_sense;            //  释放进程
        }
    else {                              //  还有进程未到达
        spin (release==local_sense) ; //  等待信号
        }
```

2. 当竞争不激烈且同步操作较少时，我们主要关心的是一个同步原语操作的延迟。
   - 即单个进程要花多长时间才完成一个同步操作。
   - 基本的旋转锁操作可在两个总线周期内完成：
     - 一个读锁
     - 一个写锁

     我们可用多种方法改进，使它在单个周期内完成操作。
3. 同步操作最严重的问题：进程进行同步操作的串行化。它大幅度地增加了完成同步操作所需要的时间。

10.5 同时多进程

线程级并行性

（Thread Level Parallelism，简称TLP）

- 线程是进程内的一个相对独立且可独立调度和指派的执行单元，它比进程要“轻巧”得多。
- 只拥有在运行过程中必不可少的一点资源，如：程序计数器、一组寄存器、堆栈等。
- 线程切换时，只需保存和设置少量寄存器的内容，开销很小。
- 线程切换只需要几个时钟周期。
- 进程的切换一般需要成百上千个处理器时钟周期。

实现多线程有两种主要的方法：

- 细粒度（fine-grained）多线程
  - 在每条指令之间都能进行线程的切换，从而使得多个线程可以交替执行。
  - 通常以时间片轮转的方法实现这样的交替执行，在轮转的过程中跳过当时处于停顿的线程。
  - CPU必须在每个时钟周期都能进行线程的切换。
  - 主要优点：既能够隐藏由长时间停顿引起的吞吐率的损失，又能够隐藏由短时间停顿带来的损失。
  - 主要缺点：减慢了单个线程的执行

➢ 粗粒度（coarse-grained）多线程

- 线程之间的切换只发生在时间较长的停顿出现时。

  例如：第二级Cache不命中。

- 减少了切换次数，也不太会降低单个线程的执行速度 。
- 缺点：减少吞吐率损失的能力有限，特别是对于较短的停顿来说更是如此。
- 原因：由粗粒度多线程的流水线建立时间的开销造成的。由于实现粗粒度多线程的CPU只执行单个线程的指令，因此当发生停顿时，流水线必须排空或暂停。停顿后切换的新线程也有个填满流水线的过程，填满后才能不断地流出指令执行结果。

10.5.1 将线程级并行转换为指令级并行

- 同时多线程技术
  - Simultaneous MultiThreading，简称SMT
  - 一种在多流出、动态调度的处理器上同时开发线程级并行和指令级并行的技术。

1. 提出SMT的主要原因

- 现代多流出处理器通常含有多个并行的功能单元，而单个线程不能有效地利用这些功能单元。
- 通过寄存器重命名和动态调度机制，来自各个独立线程的多条指令可以同时流出，而不用考虑它们之间的相互依赖关系，其相互依赖关系将通过动态调度机制得以解决。

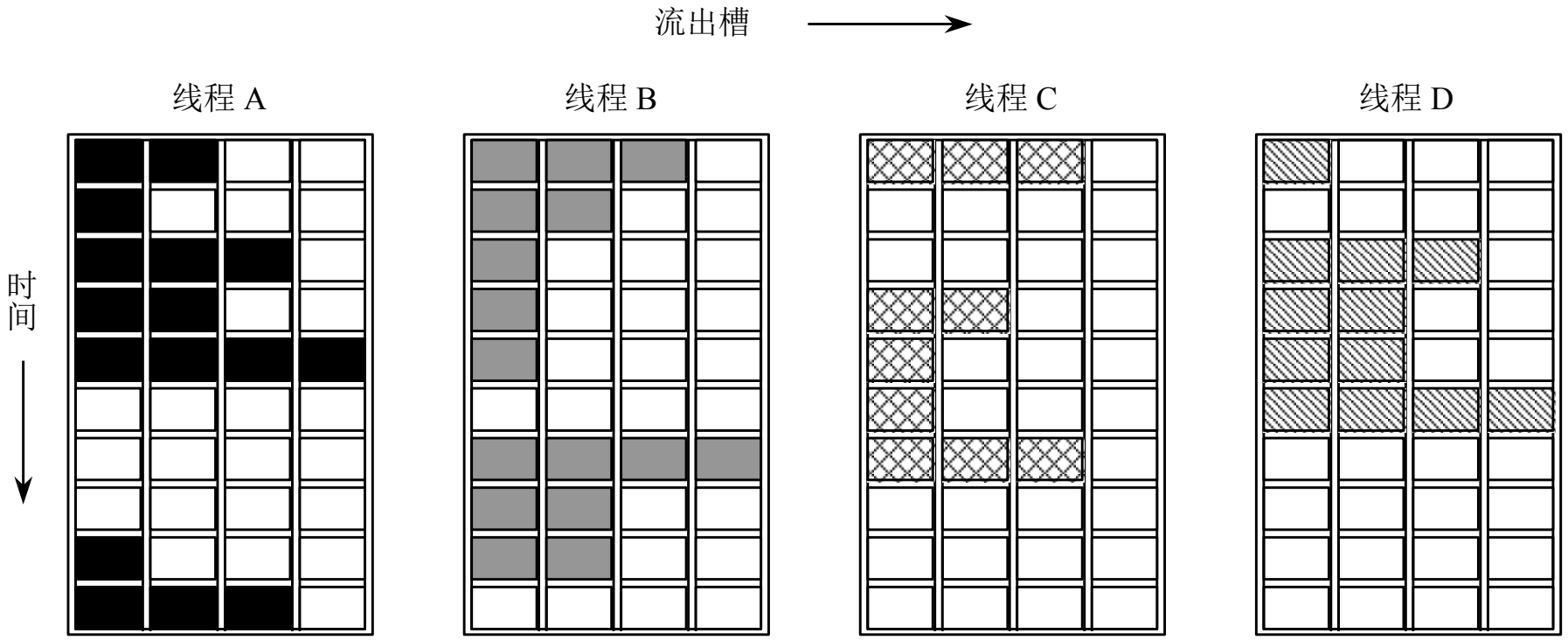
2. 一个超标量处理器在4种情况下的资源使用情况：

- 不支持多线程技术的超标量处理器

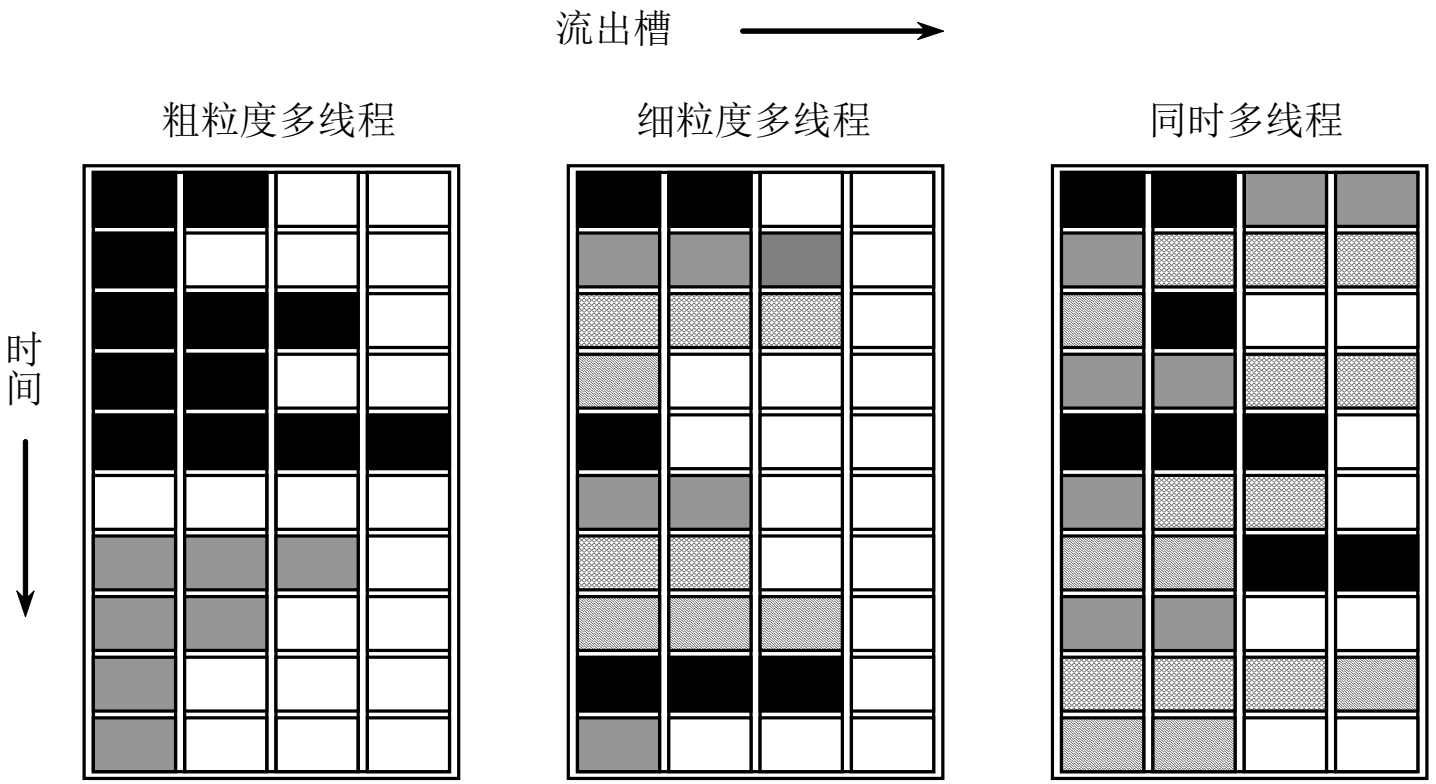
  由于缺乏足够的指令级并行而限制了流出槽的利用率。

- 支持粗粒度多线程的超标量处理器
  - 通过线程的切换部分隐藏了长时间停顿带来的开销，提高了硬件资源的利用率。
  - 只有发生停顿时才进行线程切换，而且新线程还有个启动期，所以仍然可能有一些完全空闲的时钟周期。

- 支持细粒度多线程的超标量处理器
  - 线程的交替执行消除了完全空闲的时钟周期。
  - 由于在每个时钟周期内只能流出一个线程的指令，ILP的限制导致了一些时钟周期中依然存在不少空闲流出槽。
- 支持同时多线程的超标量处理器
  - 在同一个时钟周期中可以让多个线程使用流出槽。
  - 理想情况下，流出槽的利用率只受限于多个线程对资源的需求和可用资源间的不平衡 。



不支持多线程的情况



多线程的3种情况

3. 开发的基础: 动态调度的处理器已经具备了开发线程级并行所需的许多硬件设置。
   - 动态调度超标量处理器有一组很多的虚拟寄存器，可以用作各独立线程的寄存器组。
   - 由于寄存器重命名机制给各寄存器提供了唯一的标识，多个线程的指令可以在数据路径上混合执行，而不会导致各线程之间源操作数和目的操作数的混乱。
   - 多线程可以在一个乱序执行的处理器的基础上实现，只要为每个线程设置重命名表、分别设置各自的程序计数器、并为多个线程提供指令确认的能力。

10.5.2 同时多线程处理器的设计

1. 同时多线程只有在细粒度的实现方式下才有意义
2. 细粒度调度方式会对单个线程的性能产生不利的影响

   可以通过采用优先线程的方法来尽可能地减少。这种方法既能保持多线程在性能上的优势，又对单个线程的性能影响比较少。
3. 多个线程的混合执行不可避免地会影响单个线程的执行速度

- 为提高单个线程的性能，应该为指定的优先线程尽可能多地向前取指（或许在分支指令的两条路径上都要向前取指）；
- 在分支预测失败和预取缓冲器不命中的情况下清空取指单元。但是这样限制了其他线程可用来调度的指令条数，从而降低了吞吐率。
- 所有的多线程处理器都必须在这里寻求一种折衷方案。

- 只要一有可能，处理器就运行指定的优先线程。
- 从取指阶段开始就优先处理优先线程：只要优先线程的指令预取缓冲区未满，就为它们优先取指。
- 只有当优先线程的缓冲区填满以后才为其它线程预取指令。
- 当有两个优先线程时，意味着需要并发预取两条指令流，这给取指部件和指令Cache都增添了复杂度。
- 指令流出单元也要优先考虑指定的优先线程，只有当优先线程停顿不能流出的时候才考虑其它线程。

4. 设计同时多线程处理器时面临的其它主要问题：

- 需要设置更大的寄存器组，用来保存多个线程的现场。
- 不能影响时钟周期，特别是在关键路径上

  如指令流出和指令完成：
  - 指令流出时，有更多的候选指令需要考虑；
  - 指令完成时，选择提交哪些指令可能会比较困难。
- 需要保证由于并发执行多个线程带来的Cache冲突和TLB冲突不会导致明显的性能下降。

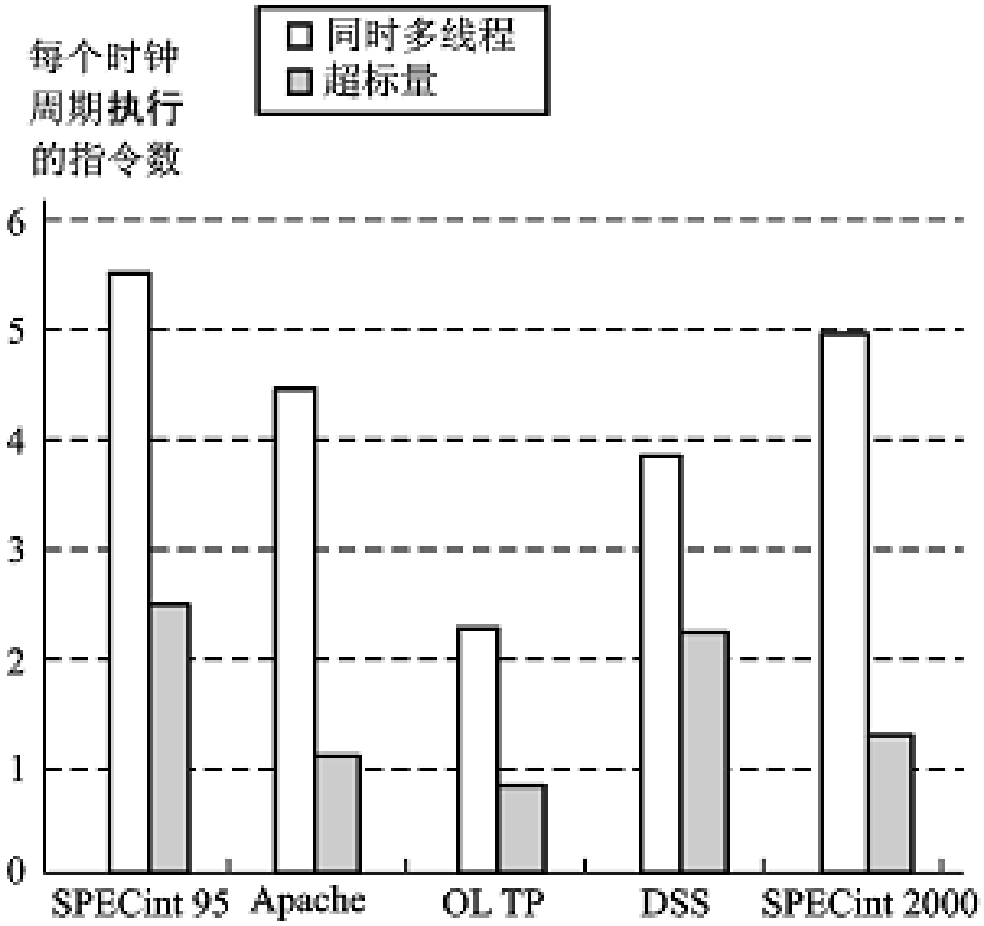
5. 需要重视的两个实际情况：

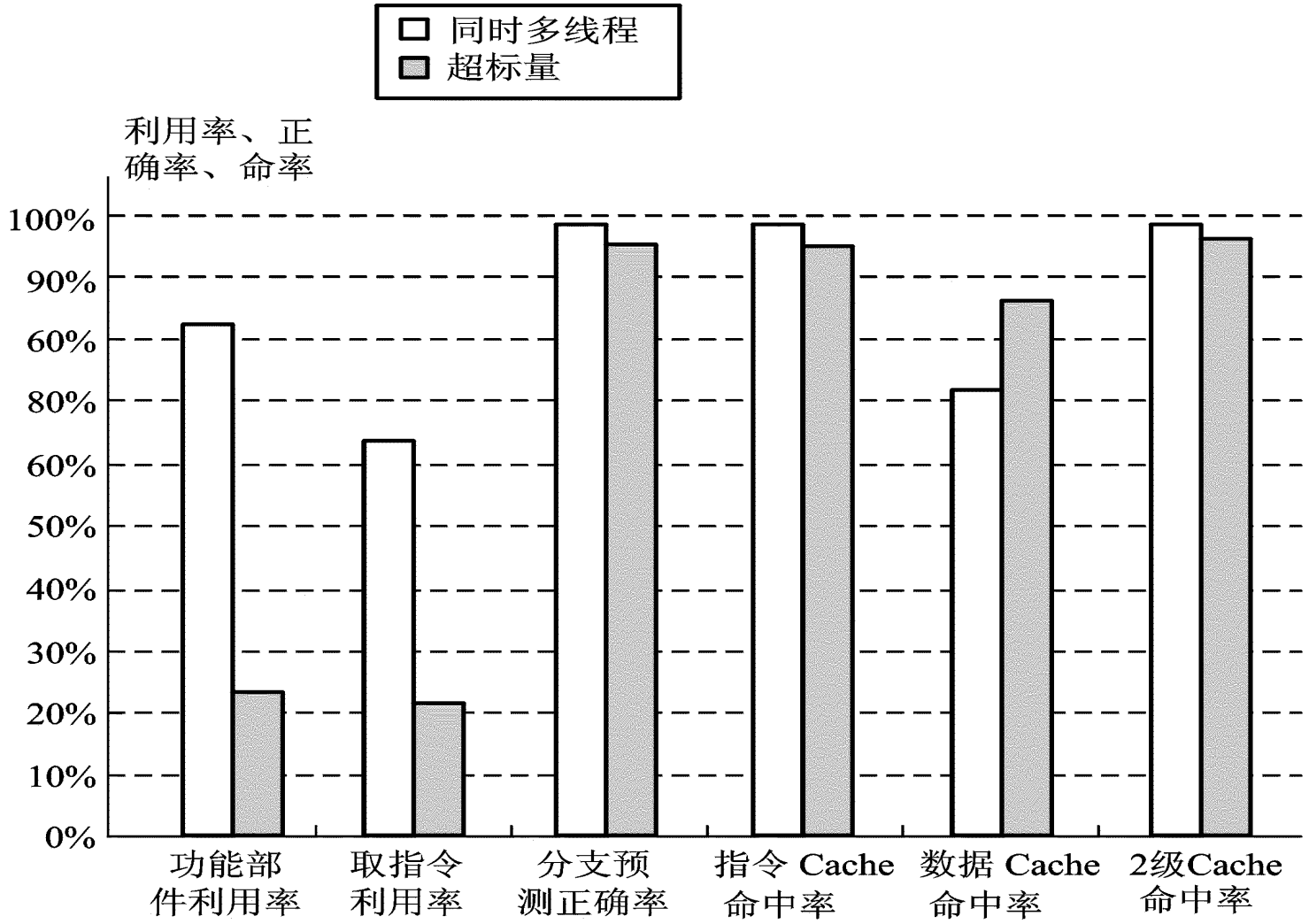
- 在许多情况下，多线程所导致的潜在额外性能开销是很小的，简单的线程切换选择算法就足够好了；
- 目前的超标量处理器的效率是比较低的，还有很大的改进余地，即使增加一些开销也是值得的。

10.5.3 同时多线程处理器的性能

在超标量处理器上增添8个线程的同时多线程能力时获得的性能提高

（单位：指令数/每拍）





SMT与基本的超标量处理器在几个主要指标上的对比

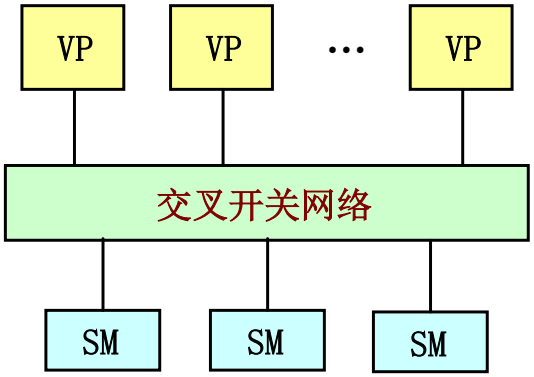
- 两个特点
  - 超标量处理器本身功能十分强大，它具有很大的一级Cache、二级Cache以及大量的功能单元。仅仅采用指令级并行，不可能利用全部的硬件性能，因此超标量处理器的设计者不可能不考虑使用诸如同时多线程这样的技术来开发线程级并行。
  - 同时多线程的能力也很强大，可以支持8个线程，并为两个线程同步取指。

将超标量和同时多线程结合起来，在指令级并行基础上进一步开发线程级并行，可以获得显著的性能提高。

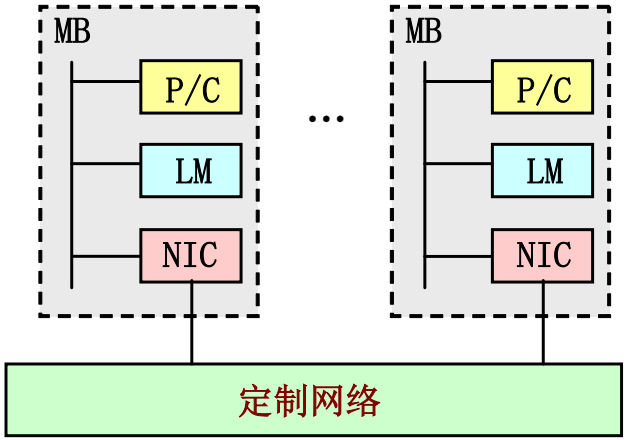
10.6 大规模并行处理机MPP

10.6.1 并行计算机系统结构

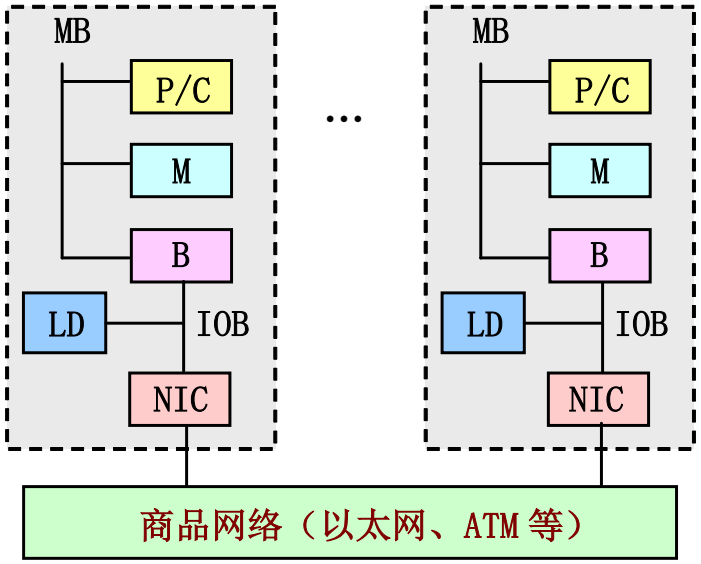
- 目前流行的高性能并行计算机系统结构通常可以分成以下5类：
  - 并行向量处理机（PVP）
  - 对称式共享存储器多处理机（SMP）
  - 分布式共享存储器多处理机（DSM）
  - 大规模并行处理机（MPP）
  - 机群计算机（Cluster）



（a）PVP



（b）MPP



（c）机群

VP：向量处理器

SM：共享存储器模块

P/C：商品微处理器/Cache

LM、M：本地存储器

NIC：网络接口电路

MB：存储器总线

LD：本地磁盘

IOB：I/O 总线

B：存储总线与 I/O 总线之间

1. 并行向量处理机
   - Cray C-90和 Cray T-90是这类机器的代表。
   - **PVP**系统一般由若干台高性能向量处理机（VP）构成。这些向量处理机是专门设计和定制的，拥有很高的向量处理性能。
   - PVP中经常采用专门设计的高带宽的交叉开关网络，把各VP与共享存储器模块SM连接起来。
   - 这样的机器通常不使用Cache，而是使用大量的向量寄存器和指令缓冲器。
2. 对称式共享存储器多处理机和分布式共享存储器多处理机

3. 大规模并行处理机

- Intel Paragon和IBM SP2是这类机器的代表。
- MPP往往是超大规模的计算机系统。
- 具有以下特点：
  - 处理结点使用商用微处理器，而且每个结点可以有多个微处理器；
  - 具有较好的可扩放性，能扩展成具有成百上千个处理器；
  - 系统中采用分布非共享的存储器，各结点有自己的地址空间；
  - 采用专门设计和定制的高性能互连网络；
  - 采用消息传递的通讯机制。

4. 机群计算机

- 一种价格低廉、易于构建、可扩放性极强的并行计算机系统。它由多台同构或异构的独立计算机通过高性能网络或局域网互连在一起，协同完成特定的并行计算任务。
  - Berkeley NOW和SP2是这类机器的代表。
- 机群的主要特点
  - 每个结点都是一台完整的计算机，拥有本地磁盘和操作系统，可以作为一个单独的计算资源供用户使用。
  - 机群的各个结点一般通过商品化网络连接在一起；
  - 网络接口以松散耦合的方式连接到结点的I/O总线。

5类机器特征比较

属性	PVP	SMP	MPP	DSM	机群
结构类型	MIMD	MIMD	MIMD	MIMD	MIMD
处理器类型	专用定制	商用	商用	商用	商用
互连网络	定制 交叉开关	总线、 交叉开关	定制网络	定制网络	商用网络 （以太网、ATM）
通信机制	共享变量	共享变量	消息传递	共享变量	消息传递
地址空间	单地址空间	单地址空间	多地址空间	单地址空间	多地址空间
系统存储器	集中共享	集中共享	分布非共享	分布共享	分布非共享
访存模型	UMA	UMA	NORMA	NUMA	NORMA
代表机器	Cray C-90， Cray T-90， NEC SX4， 银河1号	IBM R50， SGI Power Challenge， DEC Alpha 服务器8400， 曙光1号	Intel Paragon， IBM SP2， Intel TFLOPS， 曙光-1000/2000	Stanford DASH， Cray T 3D， SGI/Cray Origin 2000	Berkeley NOW， Alpha Farm， Digital Trucluster

10.6.2 大规模并行处理机MPP

1. MPP的出现和发展

- 从20世纪80年代末开始，MPP系统逐渐地显示出代替和超越向量计算多处理机系统的趋势。
  - 早期的MPP：Intel Paragon(1992年)、KSR1.Cray T3D(1993年)、IBM SP2(1994年)等。

    （分布存储的MIMD计算机）
  - MPP的高端机器：1996年Intel公司的ASCI Red

    1997年SGI Cray公司的T3E900

    （万亿次浮点运算的高性能并行计算机）

- 在这个时期，消息传递的大规模并行处理系统得到了迅速发展。

➢ 从90年代后期开始，基于消息传递的MPP系统慢慢地从主流的并行处理市场退出。

➢ 随着网络技术的发展，机群系统和MPP系统的界限越来越模糊。

- 90年代后期以来高性能计算机系统结构发展的一个趋势，新涌现的高性能计算机系统大多数都是由可扩放的高速互连网络连接的基于RISC微处理器的对称多处理机机群。
- 机群系统已经成了构建超大规模并行计算机系统的主要模式，MPP则是在慢慢地退居二三线了。

2. MPP系统概述

- MPP结构的一个重要特性：可扩放性
  - 处理器的数量可扩放至数千个处理器。
  - 主存、I/O能力和带宽也能随处理器数量的增长而成比例地增长。
- MPP主要采用了以下技术来提高系统的可扩放性。
  - 使用物理上分布的主存体系结构，使分布式主存的总容量和总带宽能随处理结点数量的增加而增加。
  - 处理能力、主存与I/O能力平衡发展。
  - 计算能力与并行性平衡发展。

3. 3种大型MPP的特点（分别代表构造大型系统的不同方法）

MPP模型	Intel/Sandia ASCI Option Red	IBM SP2	SGI/Cray Origin 2000
典型配置	9072个处理器 1.8 Tflop/s（NSL）	400个处理器 100 Gflop/s（MHPCC）	128个处理器 51 Gflop/s（NCSA）
推出日期	1996年12月	1994年9月	1996年10月
CPU类型	200 MHz，200 Mflop/s Pentium Pro	67 MHz，267 Mflop/s POWER2	200 MHz，400Mflop/s MIPS R10000
节点结构 数据存储	2个处理器， 32～256MB主存， 共享磁盘	1个处理器， 64MB～2GB本地主存， 1GB～14.5G本地磁盘	2个处理器， 64MB～256MB分布共享主存和共享磁盘
互连网络	分离二维网孔	多级网络	胖超立方体网格
访存模型	NORMA	NORMA	CC-NUMA
节点OS	轻量级内核（LWK）	完全AIX（IBM UNIX）	微内核Cellular IRIX
编程语言	基于PUMA Portals的MPI	MPI和PVM	Power C, Power Fortran
其他编程模型	NX, PVM, HPF	HPF, Linda	MPI, PVM

4. 一些典型的MPP系统的特性

结构特性	IBM SP2	Cray T3D	Cray T3E	Intel Paragon	Intel/Sandia Option Red
典型配置	400个节点 100 Gflop/s	512个节点 153 Gflop/s	512个节点 1.2 Tflop/s	400个节点 40 Gflop/s	4536个节点 1.8 Tflop/s
推出日期	1994年	1993年	1996年	1992年	1996年
CPU类型	67 MHz 267 Mflop/s POWER2	150 MHz 150 Mflop/s Alpha 21064	300 MHz 600 Mflop/s Alpha 21164	50 MHz 100 Mflop/s Intel i860	200 MHz 200 Mflop/s Pentium Pro
节点结构 数据存储	1CPU 64MB～2GB 本地存储器， 1～4.5GB 本地磁盘	2CPU 64MB主存， 50GB共享磁盘	4～8CPU 256MB～16GB DSM 主存， 共享磁盘	1～2CPU 16～128MB 本地存储器， 40GB共享磁盘	2CPU 32～256MB 本地存储器， 共享磁盘
互连网络	多级网络	三维环绕	三维环绕	二维网孔	分离二维网孔
访存模型	NORMA	NUMA	NCC—NUMA	NORMA	NORMA

结构特性	IBM SP2	Cray T3D	Cray T3E	Intel Paragon	Intel/Sandia Option Red
结构特性	IBM SP2	Cray T3D	Cray T3E	Intel Paragon	Intel/Sandia Option Red
节点OS	完全 AIX（IBM UNIX）	微内核	基于Chorus的微内核	微内核	轻量级内核（LWK）
编程模型	消息传递	共享变量、消息传递、PVM	共享变量、消息传递、PVM	消息传递	基于PUMA Portals消息传递
编程语言	MPI、PVM、HPF、Linda	MPI、HPF	MPI、HPF	NX、MPI、PVM	NX、PVM、HPF
点到点通信延迟	40μs	2μs	N/A	30μs	10μs
点到点带宽	35 MB/s	150 MB/s	480 MB/s	175 MB/s	380 MB/s

10.7 多处理机实例1：T1

1. T1的结构

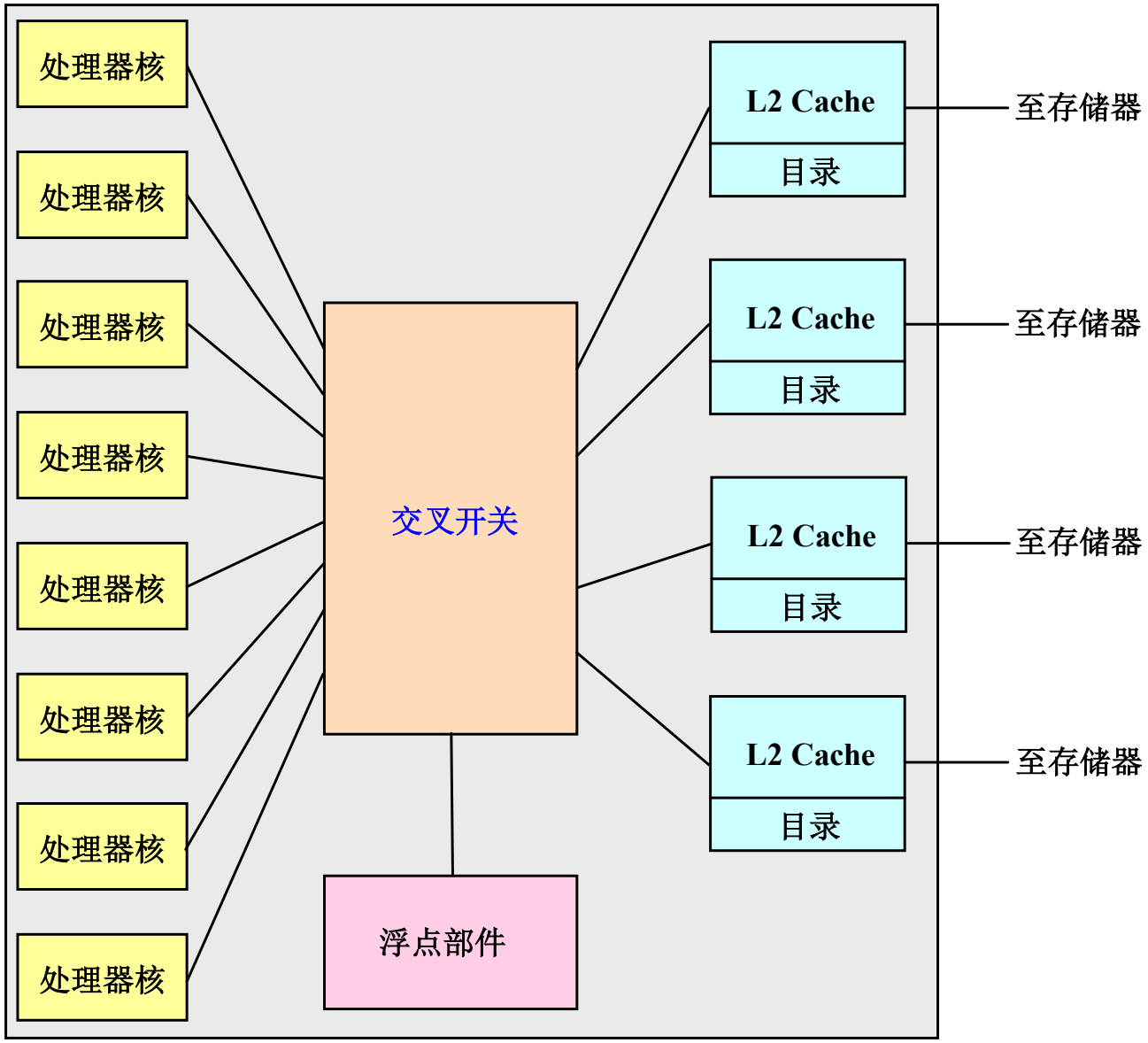
- SUN公司于2005年作为服务器处理器发布的多核多处理器。
  - 同时采用了多线程和多核技术，全面提高吞吐率。
  - 每个T1处理器中有8个处理器核，每个核最多支持4个线程。
  - 每个处理器核中都有一条6段的单流出流水线。

（类似于前面介绍的5段流水线，只是增加了一个段，用于进行线程切换。）

- 采用细粒度多线程，在每个时钟周期都可以切换到新的线程，而且在调度时可以跳过那些因流水线延迟或Cache不命中而处于等待状态的线程。
- T1处理器只有在所有其4个线程都处于等待或停顿时才会出现空闲状态。
- load和分支指令会导致3个时钟周期的延迟，不过它们都可以通过执行其他线程而被隐藏。

T1的组成结构

- 有8个核，每个核中都带有一个第一级Cache。
- 8个核通过一个交叉开关与4个第二级（L2）Cache相连。
- 用目录表法来实现Cache内容的一致性 。


处理器核
处理器核
处理器核
处理器核
处理器核
处理器核
处理器核
处理器核
交叉开关
浮点部件
L2 Cache
目录
L2 Cache
目录
L2 Cache
目录
L2 Cache
目录
至存储器
至存储器
至存储器
至存储器

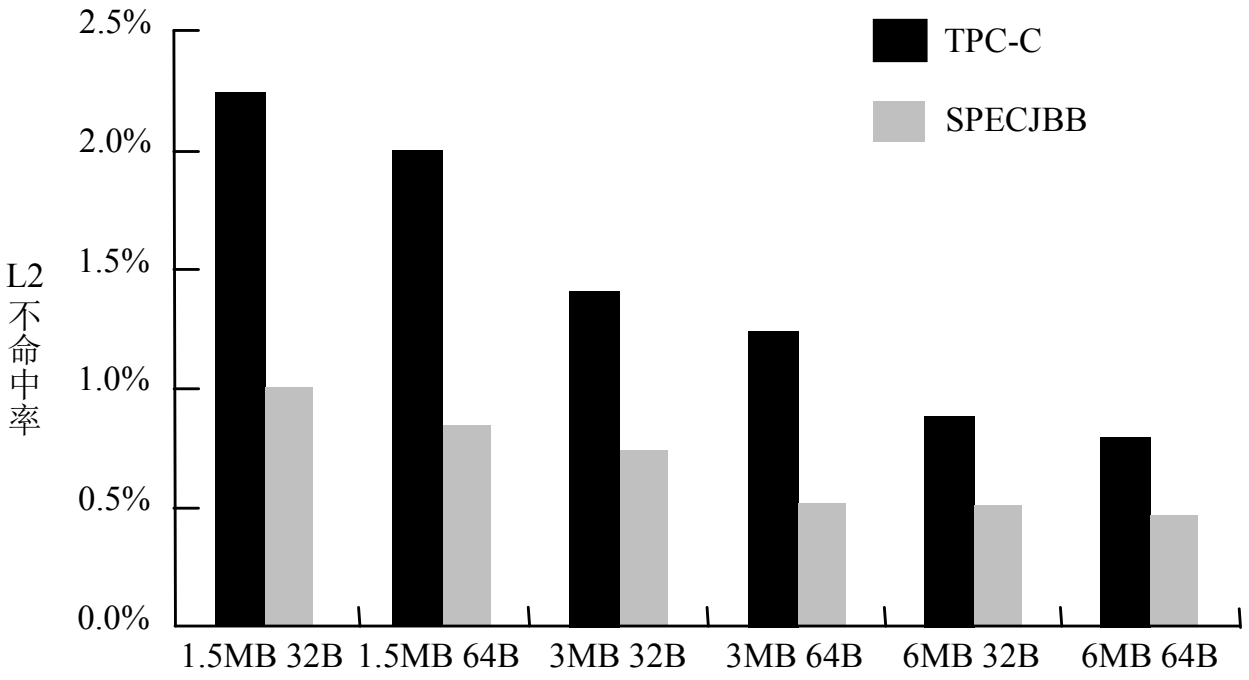
➢ T1的主要特性

特征	Sun T1
多处理器和多线程支持	每芯片8个核，每核4个线程。细粒度线程调度。 8个核共享一个浮点运算部件。支持片内多处理器。
流水线结构	简单的按序6段流水线，load和分支的延迟为3个时钟周期。
一级Cache	16KB指令Cache，8KB数据Cache。64字节块大小。 在无竞争的情况下，L1不命中的开销是23个时钟周期。
二级Cache	4个独立的二级Cache，每个750KB且和存储体相连。64字节块大小。 在无竞争的情况下，L2不命中的开销是110个时钟周期。
初始版本	90nm工艺，最高时钟频率1.2GHz，电源功率79W， 300M个晶体管，圆片面积大小379mm^2。

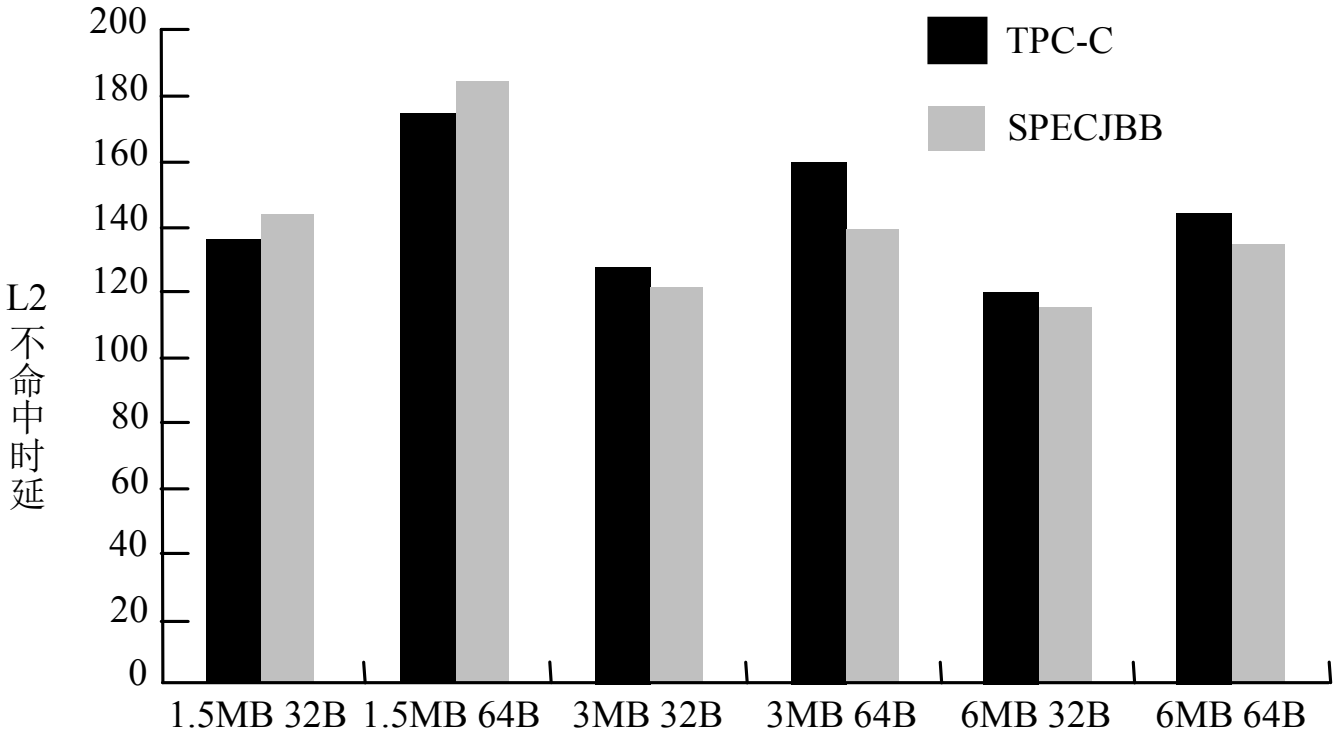
2. T1的性能

基准测试程序：TPC-C、SPECJBB、SPECWeb99

- 在以下不同情况下L2 Cache的不命中率
  - 容量分别为：1.5MB、3MB和6MB
  - 块大小分别：32B和64B



- 在不同容量和块大小的情况下（与上图同），L2 Cache的不命中延迟



- T1的每线程CPI、每核CPI以及有效的IPC（每个时钟周期完成的指令数）

基准测试程序	每线程CPI	每核CPI	8个核的有效CPI	8个核的有效IPC
TPC-C	7.2	1.8	0.225	4.4
SPECJBB	5.6	1.40	0.175	5.7
SPECWeb99	6.6	1.65	0.206	4.8

有效IPC=8÷每核CPI

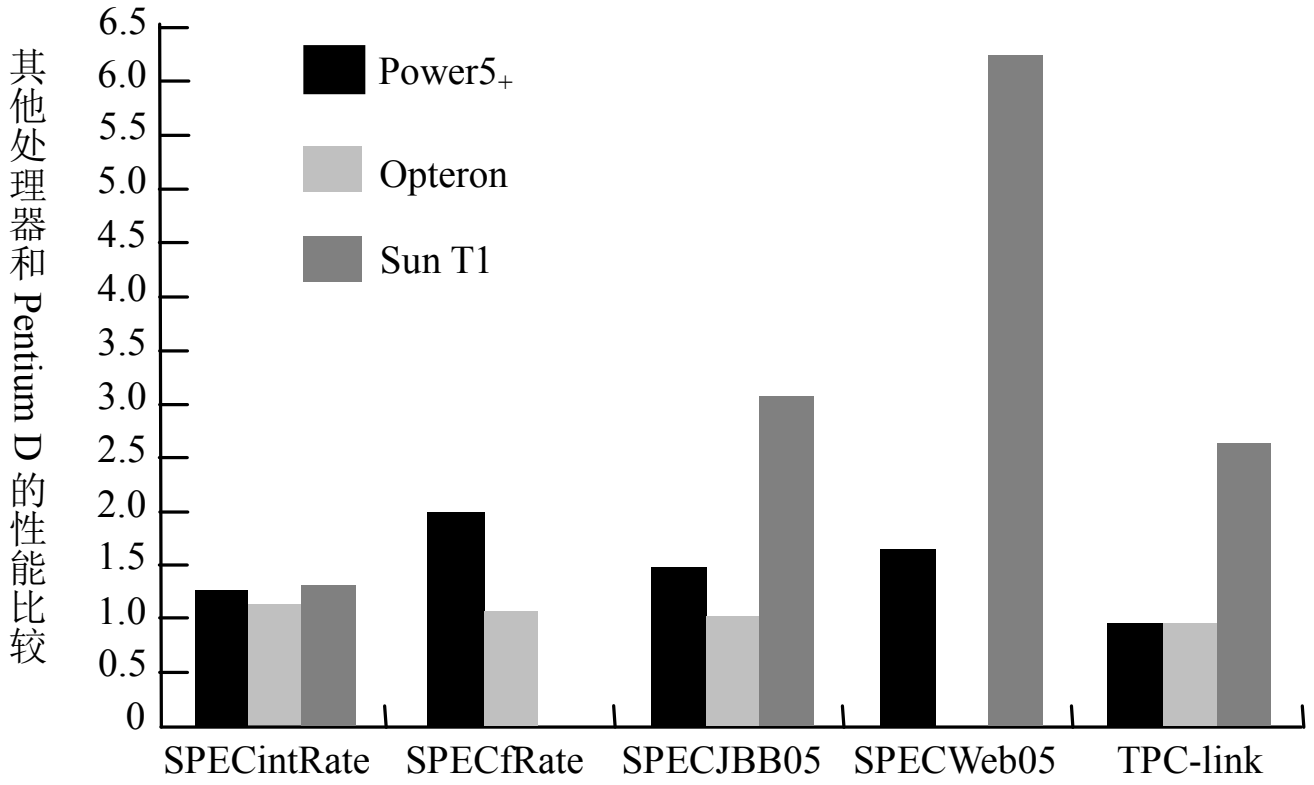
3. 4种多核处理器的性能对比

特征	SUN T1	AMD Opteron	Intel Pentium D	IBM Power 5
核	8	2	2	2
每个核每时钟周期发射的指令	1	3	3	4
多线程	Fine-grained	No	SMT	SMT
Cache	16/8	64/64	12k uops/16	64/32
一级 I/D in KB per core	3MB shared	1MB/core	1MB/core	二级： 1.9MB shared
二级 Per core/shared				三级：36MB

特征	SUN T1	AMD Opteron	Intel Pentium D	IBM Power 5
三级（off-chip）				
存储器带宽峰值（DDR2 DRAMS）	34.4GB/s	8.6GB/s	4.3GB/s	17.2GB/s
MIPS峰值	9600	7200	9600	7600
FLOPS	1200	4800(w.SSE)	6400(w.SSE)	7600
时钟频率（GHz）	1.2	2.4	3.2	1.9
晶体管数量（百万）	300	233	230	276
晶片面积（mm^2）	379	199	206	389
电源功率（W）	79	110	130	125

- 除了是重点开发ILP还是TLP的区别外，这些多核处理器还有一些根本的不同。
  - 它们在对浮点运算提供的支持以及浮点运算的性能上有很大的不同。
  - 它们的多处理器扩展能力不同，这对存储器的设计以及外部接口的使用有很大的影响。
    - Power5的可扩展性是最好的
  - 所用的实现技术差别很大，难以对它们的晶片大小和功耗进行比较。
  - 对存储器系统及其带宽的要求不同。

- 4种多核处理器的性能
  - 以SPECRate、SPECJBB2005、SPECWeb05以及类TPC-C测试基准程序为负载
  - 图中所有的数据都对Pentium D的数据进行了归一化处理，即Pentium D的值都是1。



10.8 多处理机实例2：Origin 2000

- Origin 2000系列可扩展服务器产品
  - 该系列包括：Origin 200、Origin 2000 Deskside、Origin 2000 Rack和Cray Origin 2000 4种机器。
  - Origin 2000 Deskside桌面服务器系统支持的处理器数目最多为8个
  - Origin 2000 Rack机柜服务器系统支持的处理器数目最多为16个
  - Cray Origin 2000服务器系统具有大规模扩充能力，支持的处理器数目最多可达到128个。

1. Origin 2000系列服务器产品优点
   - 不仅具有SMP的易编程和平稳扩充特性，而且还具有MPP的高可扩放性，应用非常广泛。
   - 该系列服务器综合平衡了高性能、可扩放性、可用性和兼容性，能满足许多应用的需求。
   - Origin 2000 服务器系列的I/O带宽可达102CB/s，系统传输速率比同类SMP服务器快几十倍。

     （处理、存储和传输各种多媒体信息的理想系统）
2. Origin 2000的关键技术

- CrayLink开关网络技术
  - 多重交叉开关互连技术，用于连接处理器、存储器、I/O设备等。
  - 替代总线成为处理器结点之间的互连网络。
    - 使Origin 2000 系统成为模块化系统，系统规模可以是一个基本的模块，也可以是若干模块的互连，而且还可以方便地通过增加模块数量来扩充。
    - Origin 2000 系统的可扩放性体系结构最多可以扩展至1024个处理器，而且规模增加可使系统性能也呈线性增长，包括计算能力、主存容量和带宽、系统互连带宽、I/O带宽和网络连接能力。

- 通过CrayLink，分布在所有处理器结点上的存储器在逻辑上形成单一寻址空间的共享存储器系统，但对本地和远程存储器访问的时间是不同的，是一个NUMA结构。

➢ Cellular IRIX操作系统

- 工业界最早投入使用的蜂窝式操作系统
- 将操作系统功能分布到各个处理器结点上，可以实现从小系统到大系统的无缝扩展。
  - 把多个相同的操作系统核心功能分别放到多个“蜂窝”（操作系统单元）中，每个蜂窝分别管理服务器中所有处理器的一个子集。每个操作系统单元都可以非常有效地扩展，单元之间互相通信，为用户提供一种单一的操作系统接口。

- 操作系统的这种蜂窝结构与积木式的硬件结构相结合，能够把故障隔离起来，可使故障局限于个别操作系统单元中，提高了服务器的可用性和可靠性。

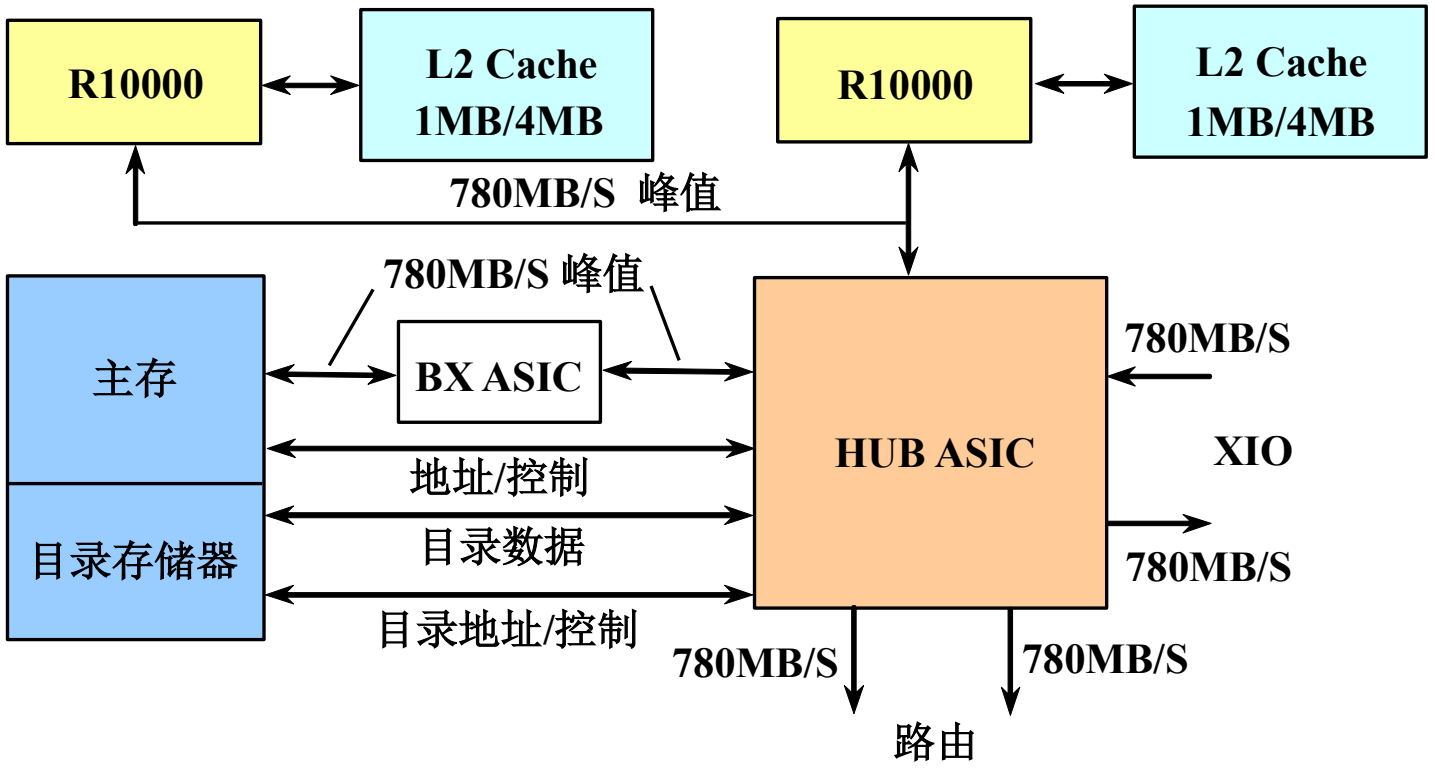
- 从SGI的IRIX演变而来的，是以UNIX为基础的64位蜂窝式操作系统。

Origin 2000系列服务器的硬件结构：

1. 结点板 （Origin200的主板 ）

➢ 组成部分

- 一个或两个MIPS R10000微处理器（内含第一级Cache）。其主频是180MHz或195MHz。



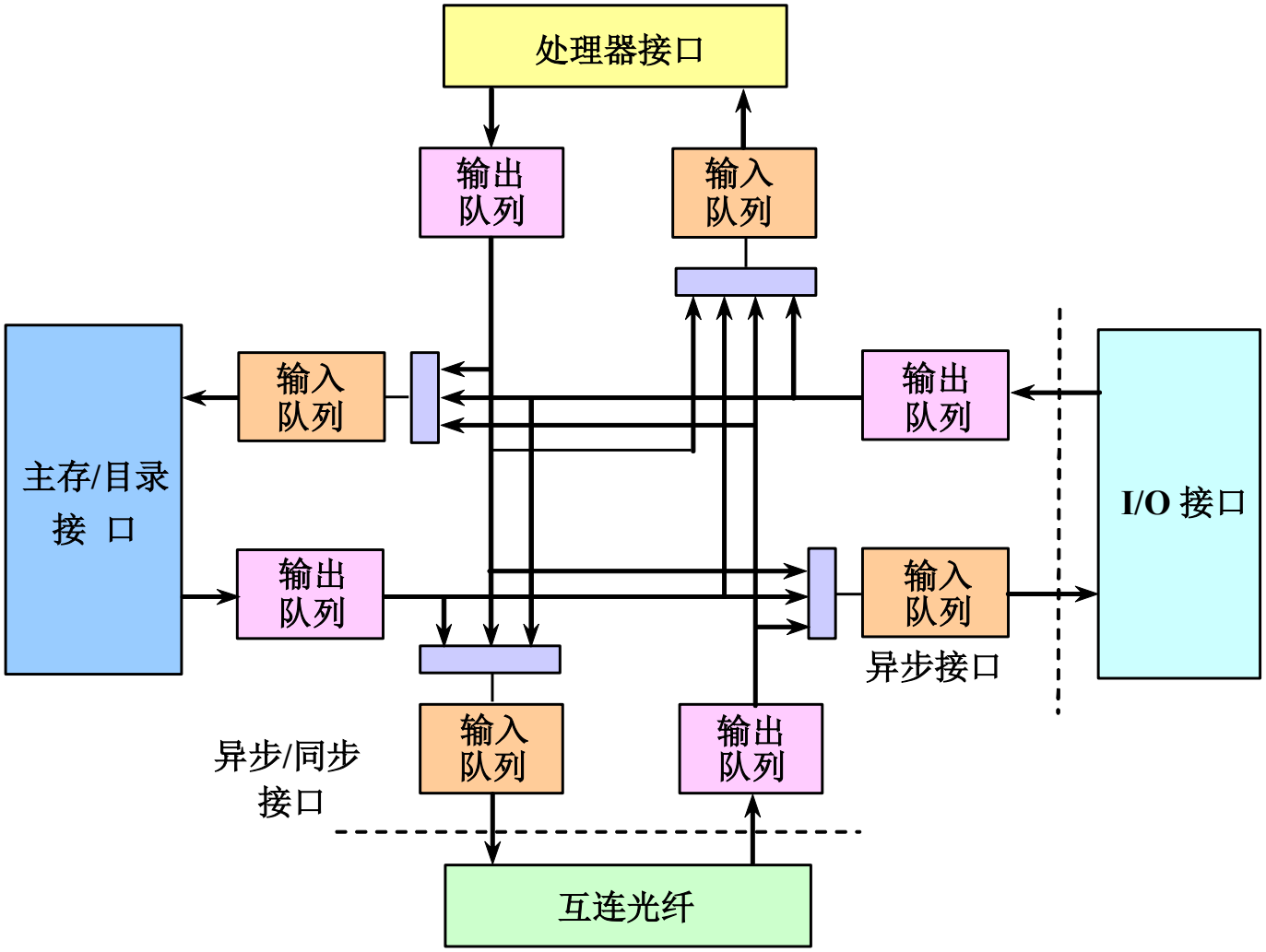
Origin 2000结点板结构

- 与处理器相配的第二级Cache，其容量为1MB或4MB；
- 主存储器（本地）以及用于实现Cache一致性的目录存储器；
- 用于实现互连的ASIC芯片，称为HUB。

  提供了4个接口：
  - 与处理器的接口
  - 与存储器的接口
  - I/O接口
  - 路由接口（接CrayLink互连网络）

➢ HUB的结构

- 4个端口在内部以交叉开关互连，通过发送消息进行通信。
- 存储器接口能双向传送数据，最大传输率为780MB/s，


处理器接口
输出队列
输入队列
输入队列
输出队列
主存/目录接 口
输出队列
输入队列
I/O 接口
异步接口
异步/同步接口
输入队列
输出队列
互连光纤

- I/O和路由器接口各有两个半双工传送端口，最大传输率为2×780MB/s，即1.56GB/s。
- 每个Hub接口连接2个先进先出（FIFO）缓冲器，分别用于输入和输出的缓冲。

2. I/O子系统

- 由一组高速链路构成。称为Crosstalk（XTALK）。
- Crosstalk I/O系统是分布的，在每个结点板上有一个I/O端口，可以被每个处理器访问。
- I/O操作通过结点板上的单端口Crosstalk协议的链路进行控制，或者通过在Crossbow（XBOW）

ASIC芯片上的智能交叉开关进行互连。

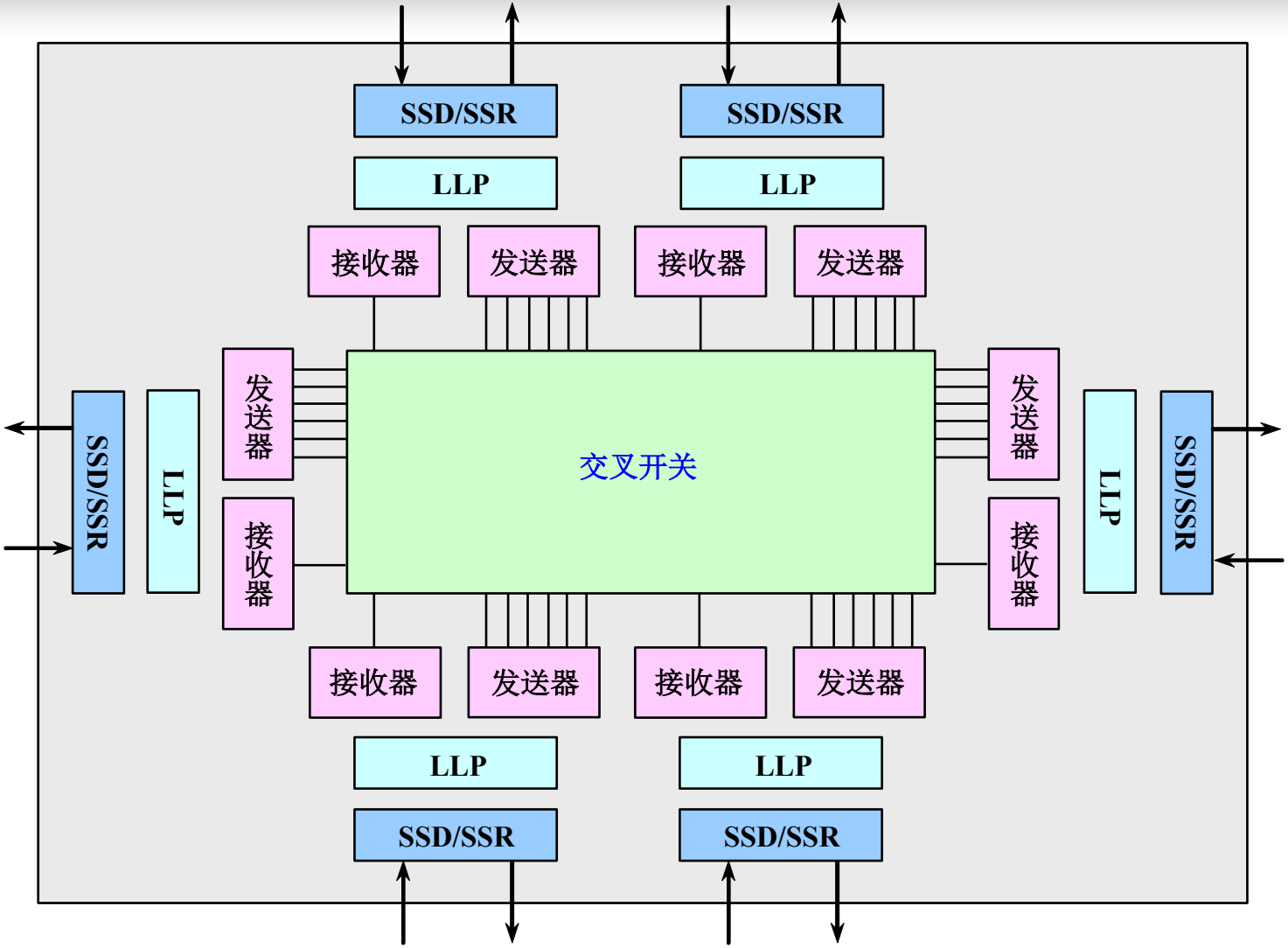
- XBOW ASIC芯片将Crosstalk I/O端口扩充到8个端口。
  - 6个端口用于I/O
  - 2个端口用于连接到结点板

3. 互连网络子系统

- 互连网络子系统是由路由器和链路构成的。
  - 每个路由器由一组交叉开关组成，能实现多路无阻塞连接。
  - 每条双向链路带宽峰值达到1.6GB/s。

- 互连网络CrayLink Interconnect为每对结点提供至少两条独立链路进行通信。
  - 这种结构使得结点之间的通信可以绕过不能运行的路由器和断开了的链路。
- 路由器将结点板上的HUB物理地连接到CrayLink Interconnect上。
  - 路由器的核心：实现6路无阻塞交叉开关的路由ASIC芯片
  - 路由器的交叉开关允许6个路由端口全双工同时操作，每个端口有2条单向的数据通路。
- 路由ASIC芯片的结构

10.8 多处理机实例2：Origin 2000

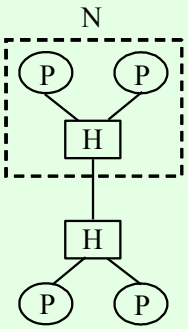


- ASIC芯片的主要功能
  - 选择发送端口和接收端口的最高效连接，动态地切换6个端口的连接。
  - 在CrayLink Interconnect的链路层协议（LLP）控制下与其他路由器和HUB进行可靠通信；
  - 消息的包以虫蚀寻径方式通过路由器以减少通信时延；
  - 对CrayLink信息提供缓存。
- 路由器提供的峰值通信带宽达到9.36GB/s。

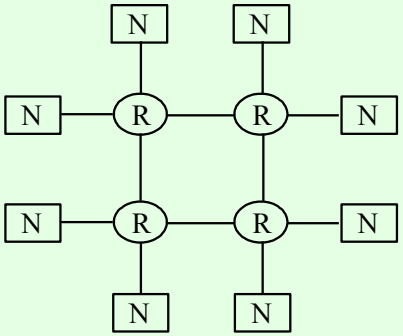
4. 不同的配置和互连

- Origin 2000在不同处理器个数配置情况下的互连拓扑结构
  - 处理器数目：4、16、32、64和128个
  - P：处理器
  - N：结点板
  - H：HUB
  - R：路由器
- 128处理器系统

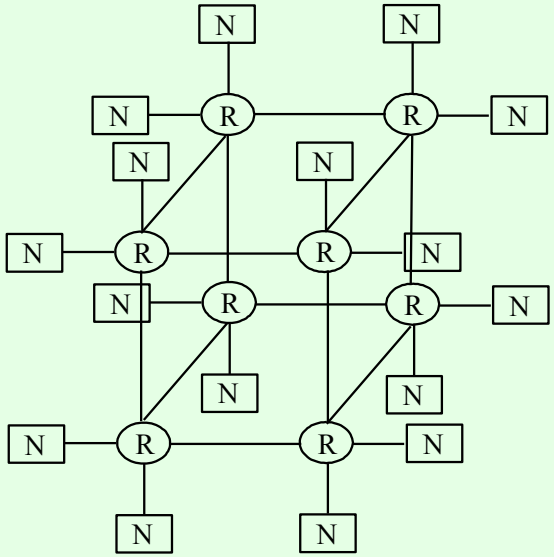
  由4个立方体组成，在立方体之间传送数据多经过了一级路由器。



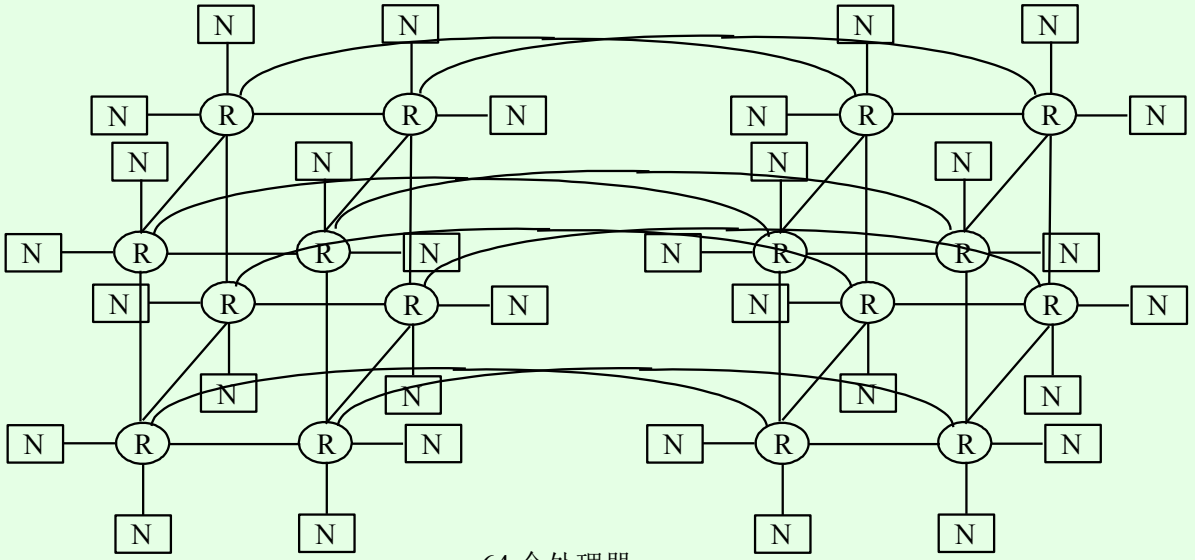
4 个处理器



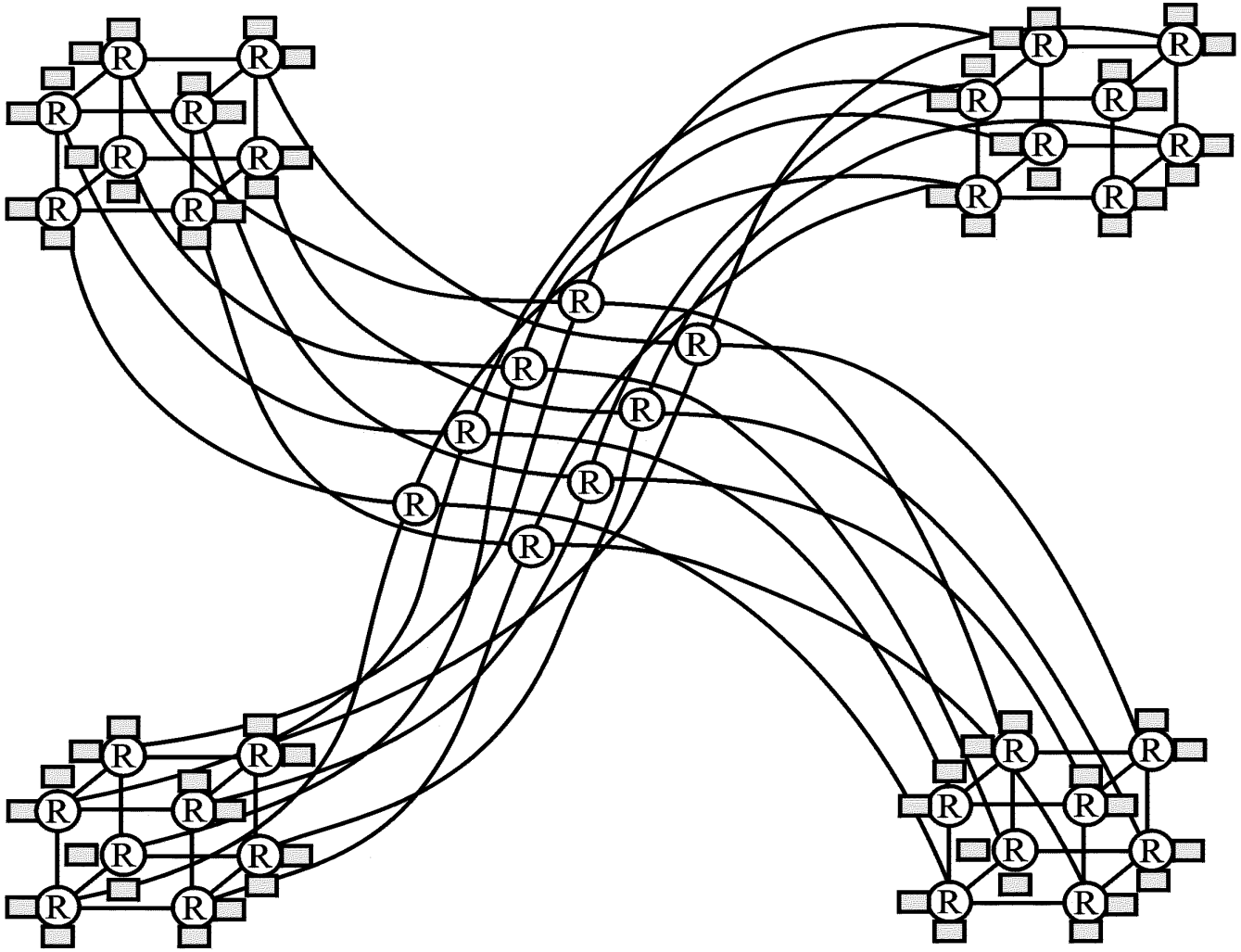
16 个处理器



32 个处理器



64 个处理器



128个处理器

- 在结点内部实现的是SMP（对称多处理器）结构，由于只有两个处理器，所以不存在SMP结构的总线瓶颈问题。
- 在结点之间实现的是大规模并行处理结构，但又解决了共享存储器问题。因此在Origin系统中，无论是访问存储器的时间还是结点间传送数据的带宽都很理想。

5. Origin系统中CPU访问存储器的延迟时间

- 假设：
  - CPU的主频为195MHz
  - Cache不命中
  - 最小延迟时间：CPU访问本结点存储器的时间
  - 最大延迟时间：CPU访问距离最远的存储器的时间

Origin系统中CPU访问存储器的延迟时间

系统CPU数	最小延迟时间	最大延迟时间	平均延迟时间
2	318ns	343ns	343ns
4	318ns	554ns	441ns
8	318ns	759ns	623ns
16	318ns	759ns	691ns
32	318ns	836ns	764ns
64	318ns	1067ns	851ns
128	318ns	1169ns	959ns

6. Origin系统的带宽

每个Hub连到路由器和互连网络的最大频宽为：

1.56Gb/s（全双工，2×780Mb/s）

系统处理器数	带宽 （无快速传送连线）	带宽 （无快速传送连线）
8	1.56Gb/s	3.12Gb/s
16	3.12Gb/s	6.24Gb/s
32	6.24Gb/s	12.5Gb/s
64	12.5Gb/s	--
128	25Gb/s	--

7. 存储层次

寄存器、L1 Cache、L2 Cache和主存储器

- 寄存器和L1 Cache在R10000微处理器中
- 寄存器的存取时间最短
- L1 Cache又分成指令Cache和数据Cache两部分

  （避免取指令和存/取数据发生冲突）
- L2 Cache安装在结点卡中，统一存放指令和数据，由SRAM组成。
- 主存储器地址是统一编址的，每个处理器通过互连网络可访问系统中任一存储单元。

8. 实现Cache的一致性

- 基于目录协议与写作废协议
- 每个结点中，有一个存储器和一个目录存储器。
- 每块对应于一个目录项，每个目录项包含其对应存储器块的状态信息和系统中各Cache共享该存储块情况的位向量，根据位向量可以知道哪些Cache中有其副本。