北京郵電大學
Beijing University of Posts and Telecommunications

Computer Architecture

计算机系统结构

(Multiprocessors/Multicomputers)

北京邮电大学
邝 坚 2020年5月

多处理机

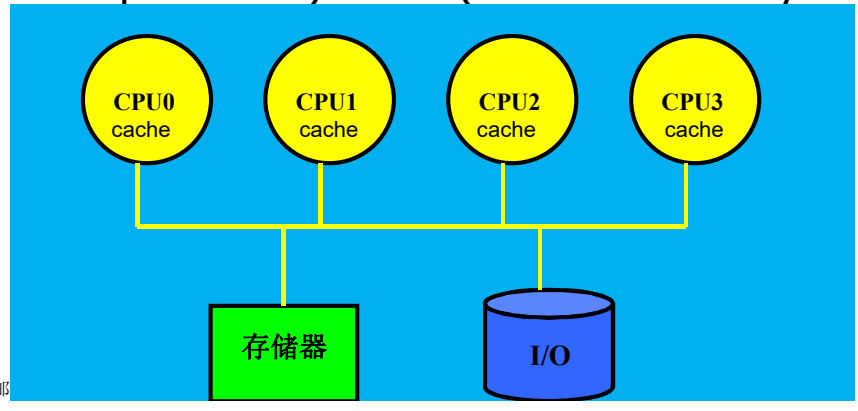
- 大幅度提高单处理机性能受到较大挑战，虽然未走到尽头
- 多处理机正起着越来越重要的作用
  - Intel于2004年宣布放弃了其高性能单处理器项目，转向多核（multi-core）的研究和开发。其他半导体公司也纷纷走上此途径
  - 并行计算机应用软件已有了稳定的发展。
  - 充分利用商品化微处理器所具有的高性能价格比的优势。
- 多处理机设计的主流：中小规模的计算机（处理器的个数＜32，本章重点）

MIMD

- 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。
- 根据存储器的组织结构，把现有的MIMD机器分为两类
  - 集中式共享存储器结构
  - 分布式存储器多处理机

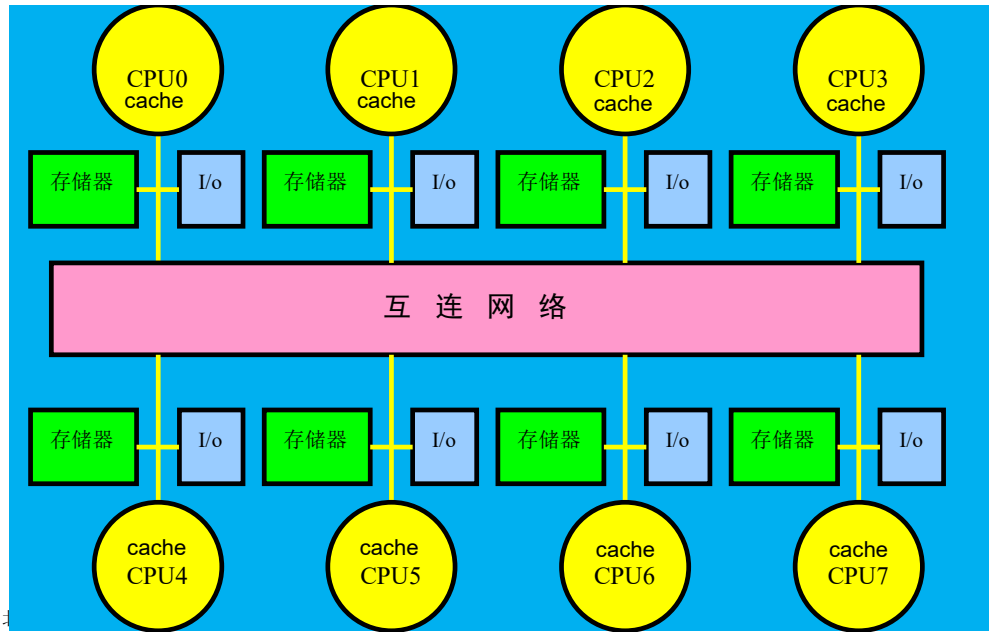
集中式共享存储器结构

- Centralized Shared-Memory Architecture
  - 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对称式共享存储器多处理机(SMP - Symmetric shared-memory Multiprocessor)，UMA(Uniform Memory Access)



分布式存储器多处理机

- 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



分布式存储器多处理机

- 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。

两种存储器系统结构

- 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(NUMA - Non-Uniform Memory Access)

两种存储器系统结构

- 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

两种通信机制

- 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。

消息传递通信机制

- 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。发送方发送成功即可继续

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。

不同通信机制的优点

- 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。

两种通信机制

- 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。

并行处理面临的挑战

- 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
- 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：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：

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

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

由上式可得：并行比例＝0.9975

并行处理面临的挑战

- 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

举例

- 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，假设所有其他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?

举例

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（如Cache缓冲共享数据）和编程技术（重新组织数据，形成更多的局部访问）

并行处理面临的挑战

- 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
- 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／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。

对称式共享存储器系统结构

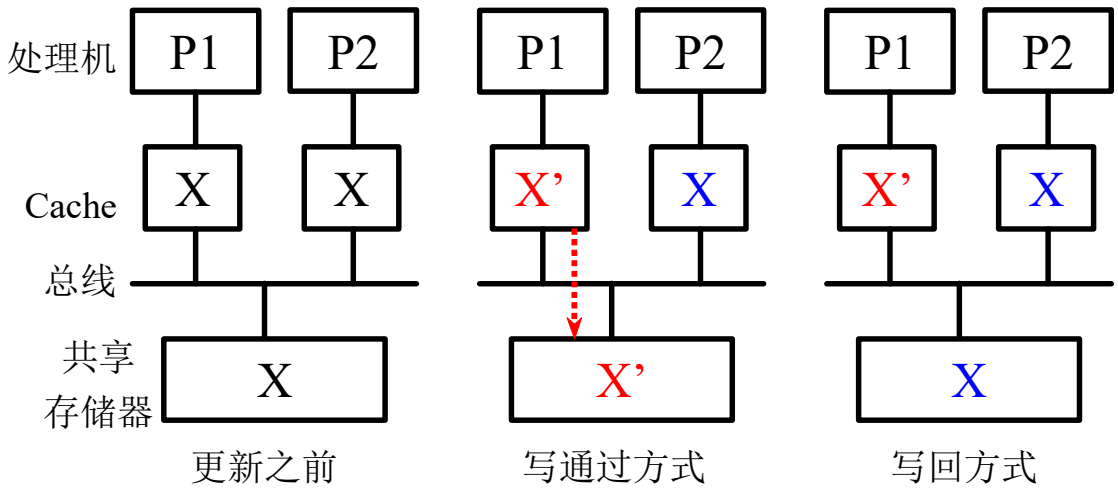
- 多个处理器共享一个存储器。
- 当处理机规模较小时，这种计算机十分经济
- 近些年，能在一个单独的芯片上实现2～8个处理器核。
  - 例如：Sun公司　2006年　T1　8核的多处理器
- 支持对共享数据和私有数据的Cache缓存
  - 私有数据供一个单独的处理器使用，而共享数据则是供多个处理器使用。
- 共享数据进入Cache产生了一个新的问题
  - Cache的一致性问题

Cache coherence

- 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。
- 出现不一致性问题的原因举例：
  - 共享可写的数据
  - 进程迁移
  - I/O传输。

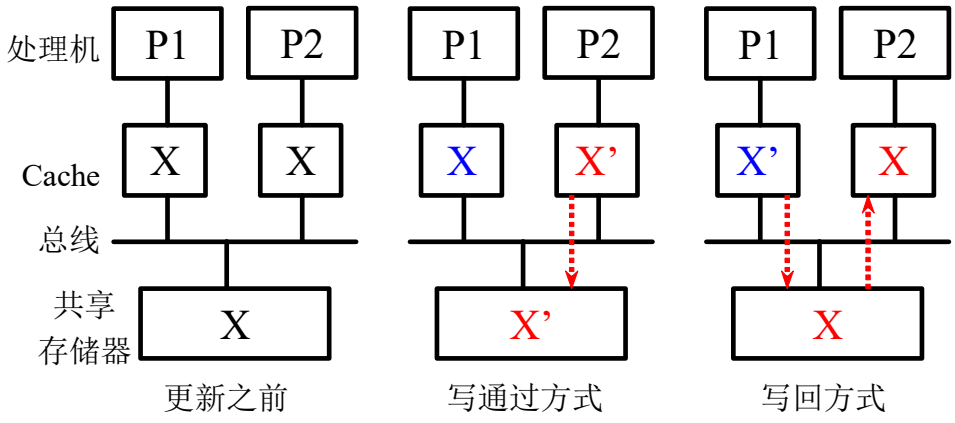
写共享数据引起的不一致性

- 使用多个局部Cache时，可能发生Cache不一致性问题：



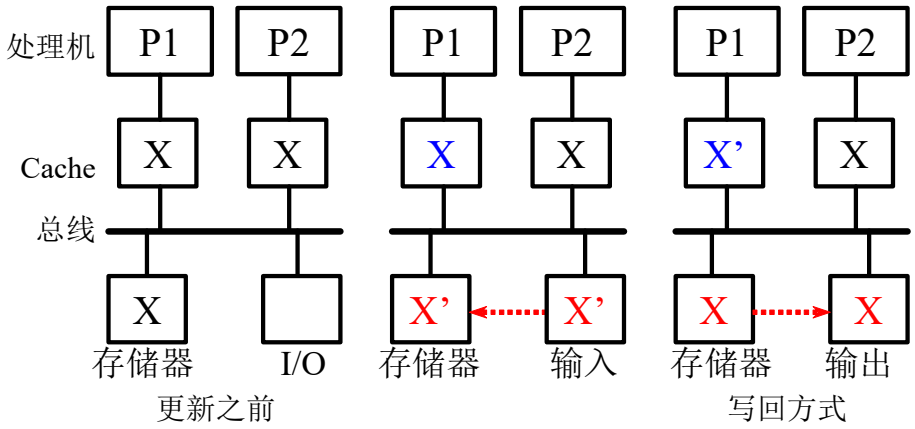
进程迁移引起的数据不一致性

- P1和P2中都有X的拷贝，P2修改X (写通过) 。进程迁移到P1上，这时P1的Cache中仍然是X。
- P1中有X的拷贝，P2中没有，P1修改X(写回) 。进程迁移到了P2上，P2运行时从内存中读到是X。



I/O造成数据不一致性

- I/O使用DMA操作



Cache coherence

- 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

Cache coherence

- 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，读和写之间没有其他处理器对单元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，读和写之间无其他写，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)

实现一致性的基本方案

- 在支持Cache一致性的多处理机中，Cache提供两种功能：
  - 共享数据的迁移(Migration) – 将远程的共享数据拷贝迁入本地Cache
    - 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(Replication) – 把多个处理器需要同时读取的共享数据在这些处理器本地Cache中各存一个副本
    - 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。

实现一致性的基本方案

- 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。
    - 可用于较大规模的多处理机，但开销较大

实现一致性的基本方案

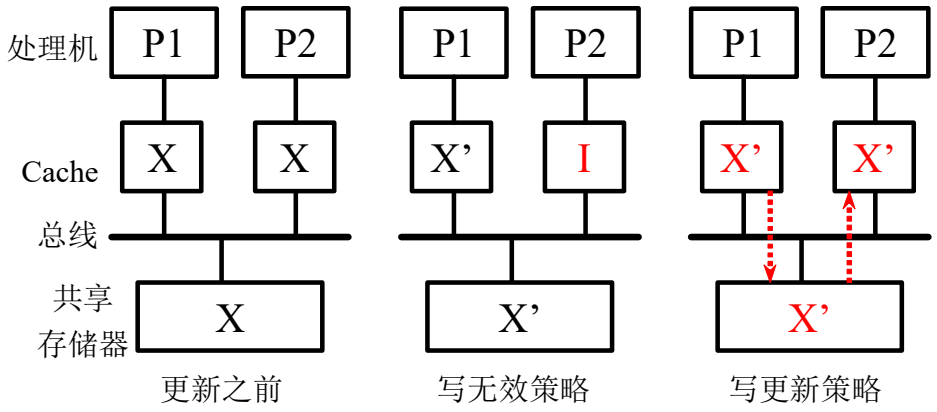
- 监听式协议（snooping）
  - 每个Cache除了包含物理存储器中块的数据拷贝之外，也保存着各个块的共享状态信息。
  - Cache通常连在共享存储器的总线上，当某个Cache需要访问存储器时，它会把请求放到总线上广播出去，其他各个Cache控制器通过监听总线（它们一直在监听）来判断它们是否有总线上请求的数据块。如果有，就进行相应的操作。
  - 多个处理器中每个Cache都与单一共享存储器相连，一般都采用监听式协议

邝坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

监听协议 - Snooping Protocol

- 利用总线具有的广播能力，用分散控制的办法解决Cache一致性问题
- 方法一：写无效/写作废（Write Invalidate）策略，在本地Cache的数据块修改时使远程数据块都无效/作废
- 方法二：写更新（Write Update）策略，在本地Cache数据块修改时通过总线把新的数据块广播给含该数据块的所有其他Cache

监听协议 - Snooping Protocol



- 在Cache控制器中要增加相应的控制线路
- 要占用总线时间，只能用于处理机数量不多的多处理机系统中
- MESI - Modified/Exclusive/Shared/Invalid

监听协议 - Snooping Protocol

- 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。

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：
  - 采用某种方法保证对同一个Cache Block的写访问串行化
  - 如果由总线实现，即获取总线控制权的顺序性

监听协议的基本实现技术

- 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。

监听协议的基本实现技术

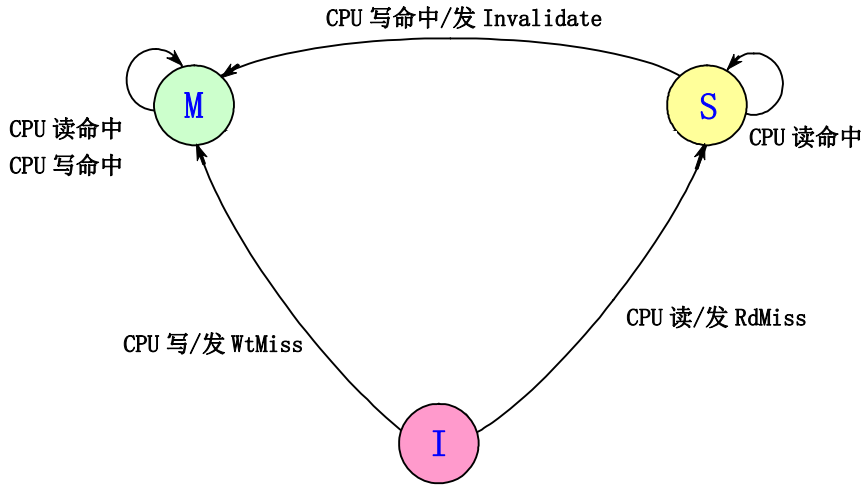
- 有的监听协议还增设了一条Invalidate消息，用来通知其他各处理器作废其Cache中相应的副本。
  - 与WtMiss的区别：Invalidate不引起调块
- Cache的标识（tag）可直接用来实现监听。
- 作废一个块只需将其有效位置为无效。
- 对于写操作，如果知道其他处理器无该数据的副本，就可以简化操作 - 给每个Cache块增设一个共享位
  - 为“1”：该块是被多个处理器所共享
  - 为“0”：仅被某个处理器所独占

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（I）：Cache中该块的内容为无效。
  - 共享（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（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

监听协议举例

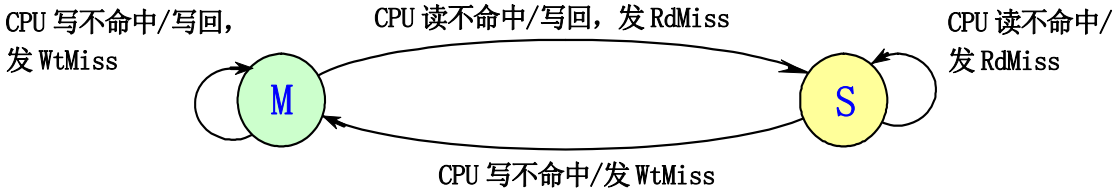
- 响应来自处理器的请求（Local）
  - 不发生替换的情况



写作废协议中（采用写回法），Cache块的状态转换图

监听协议举例

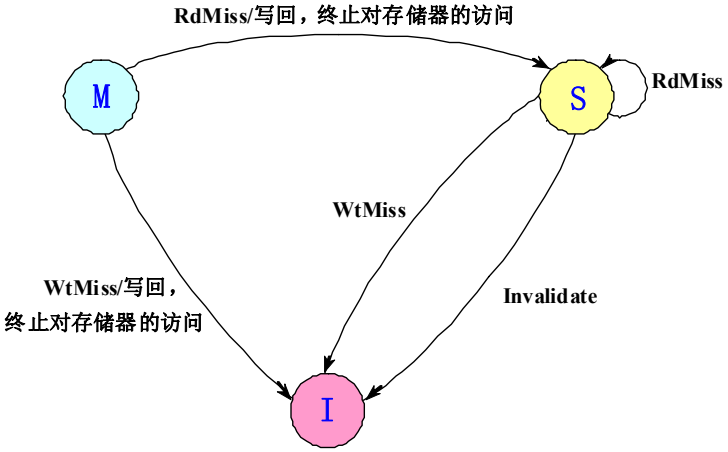
- 发生替换的情况



写作废协议中（采用写回法），Cache块的状态转换图

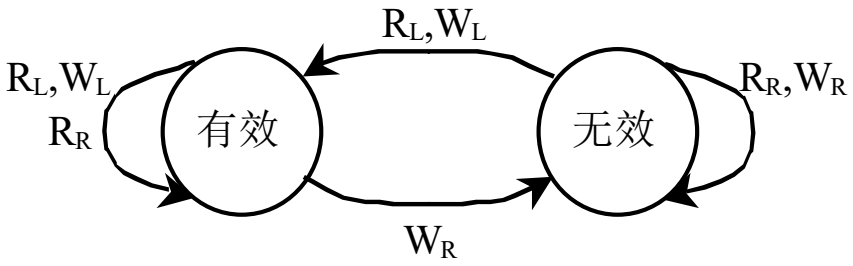
监听协议举例

- 响应来自总线的请求（Remote）
  - 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

MESI - WriteThrough



目录表(directory-based)协议

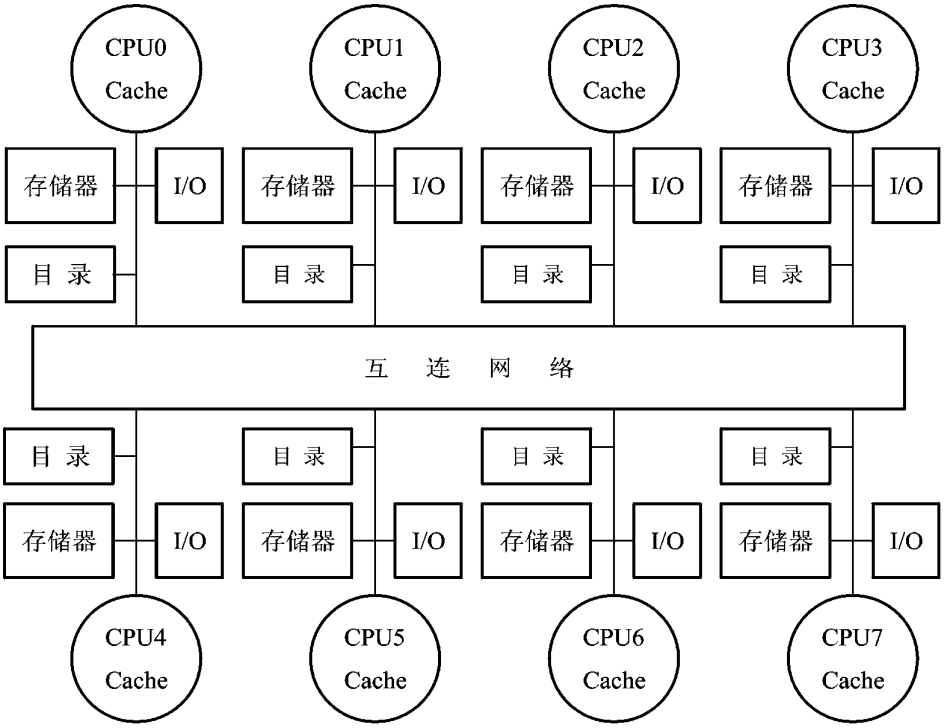
- 在非总线结构的多处理机系统中，采用基于目录的Cache一致性协议
  - 目录协议的基本思想
    - 广播和监听的机制使得监听一致性协议的可扩放性很差。
    - 寻找替代监听协议的一致性协议。
  - 目录：一种集中的数据结构。对于存储器中的每一个可以调入Cache的数据块，在目录中设置一条目录项，用于记录该块的状态以及哪些Cache中有副本等相关信息。
  - 特点：对于任何一个数据块，都可以快速地在唯一的一个位置中找到相关的信息。这使一致性协议避免了广播操作。

目录表(directory-based)协议

- 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



目录表(directory-based)协议

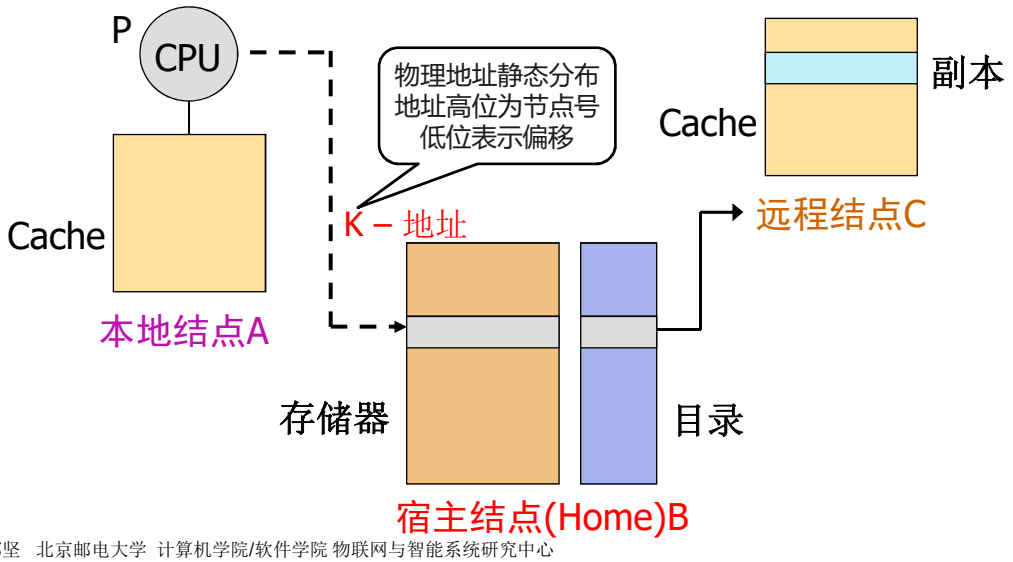
- 目录法最简单的实现方案：对于存储器中每一块都在目录中设置一项。目录中的信息量与M×N成正比。其中：
  - M：存储器中存储块的总数量
  - N：处理器的个数
  - 由于M=K×N， K是每个处理机中存储块的数量，所以如果K保持不变，则目录中的信息量为$K\times N^2$，就与N^2成正比。
  - 因此此方法适合于处理器数N较小的情况。

目录表(directory-based)协议

- 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。

目录表(directory-based)协议

- 本地结点、宿主结点以及远程结点的关系
  - 宿主结点：包含所访问的存储单元及其目录项的结点；
  - 本地结点：发出访问请求的结点，可能同时是宿主节点；
  - 远程结点：可以和宿主结点是同一个结点，也可以不同。



目录表(directory-based)协议

- 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

目录表(directory-based)协议

- 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K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K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。

目录表(directory-based)协议

- 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。

目录表(directory-based)协议

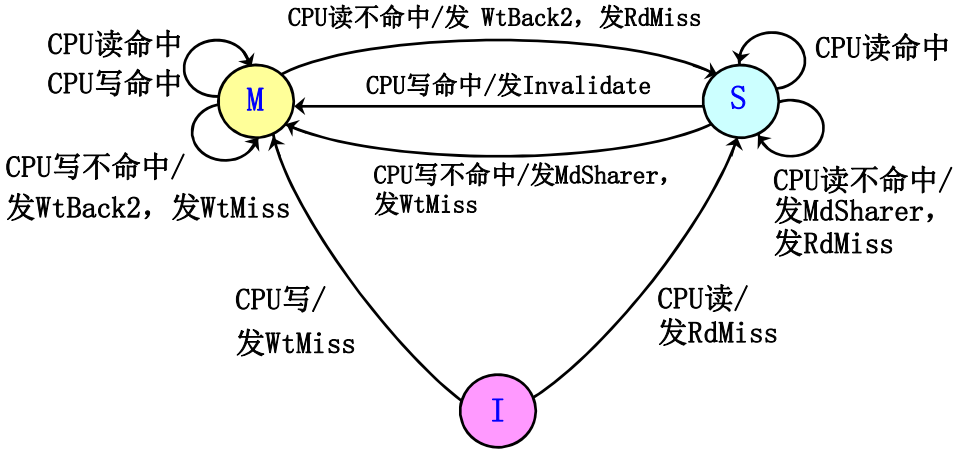
- 宿主结点发送给本地结点的消息
  - DReply（D）
    - D表示数据内容。把从宿主存储器获得的数据返回给本地Cache。
- 远程结点发送给宿主结点的消息
  - WtBack（K，D）
    - 把远程Cache中包含地址K的数据块写回到宿主结点中，该消息是远程结点对宿主结点发来的“取数据”或“取/作废”消息的响应。

目录表(directory-based)协议

- 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。

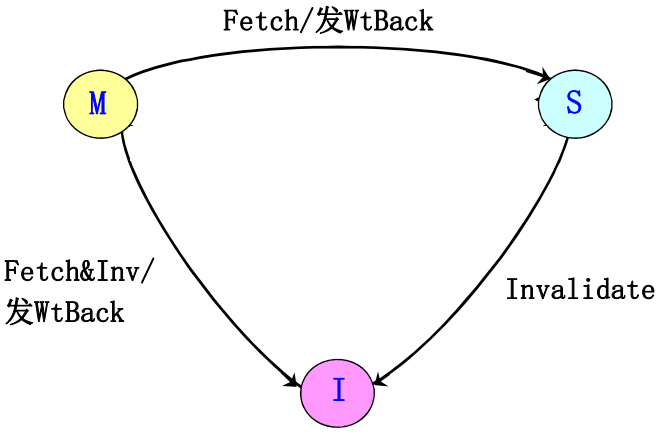
目录表协议实例

- 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



目录表协议实例

- 远程结点中Cache块响应来自宿主结点的请求的状态转换图

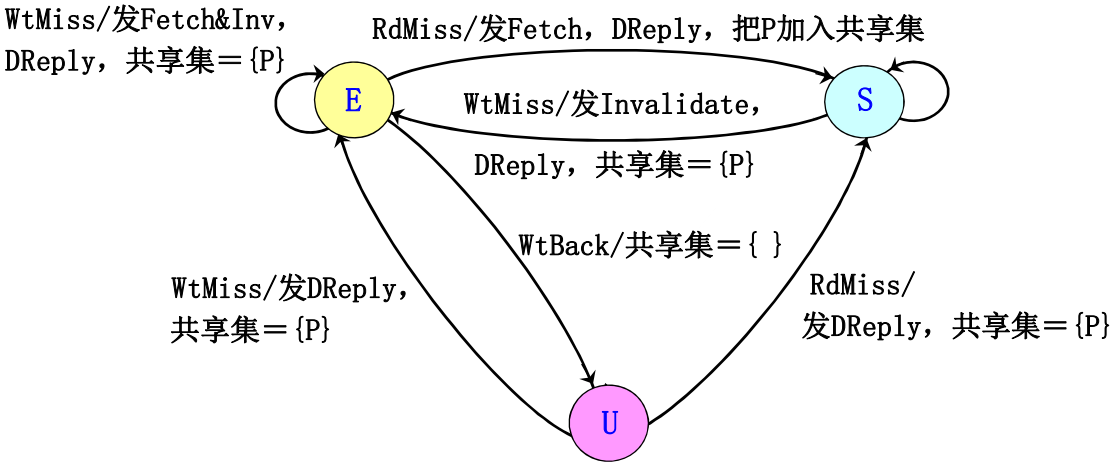


目录表协议实例

- 目录的状态转换及相应的操作
  - 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，目录所进行的操作有：
    - 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（假设这些操作是原子的）
    - 读不命中
    - 写不命中
    - 数据写回

目录表协议实例

- 目录的状态转换及相应的操作



U：未缓存（Uncached）　　S：共享（Shared）：只读

E：独占（Exclusive）：可读写　　P：本地处理器

目录表(directory-based)协议

- 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。

目录表(directory-based)协议

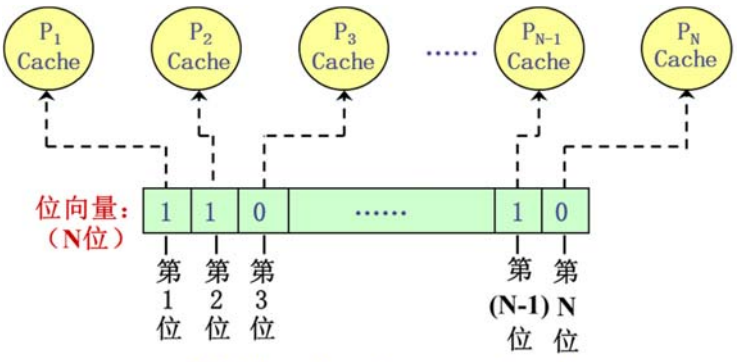
- 当一个块处于共享状态时，其在存储器中的数据是当前最新的，对该块发出的请求及其处理操作为：
  - RdMiss
    - 将存储器数据送往请求方处理机，并将其加入共享集合。
  - WtMiss
    - 将数据送往请求方处理机，对共享集合中所有的处理机发送作废消息，且将共享集合改为仅含有该处理机，该块的状态变为独占。

目录表(directory-based)协议

- 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
    - 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。
    - 该块将有一个新的拥有者，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。

目录表结构

- 目录的三种结构 - 全映像，有限映像，链式映像
- Full-Map directory
  - 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。
  - 当位向量中的值为“1”时，表示所对应的处理机有该数据块的副本。
  - 共享集合由位向量中值为“1”的位所对应的处理机构成。

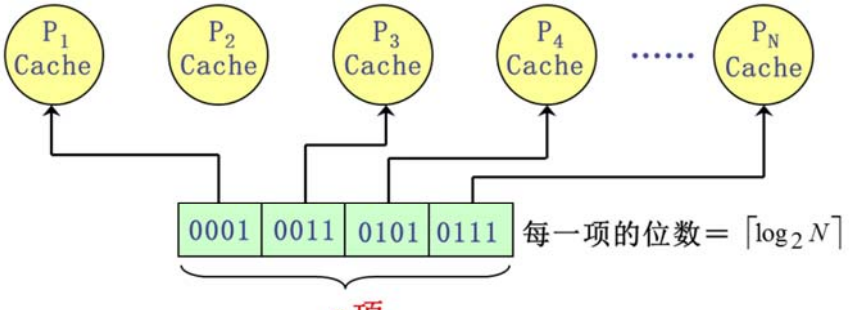


Full-Map directory

- 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。

Limited directory

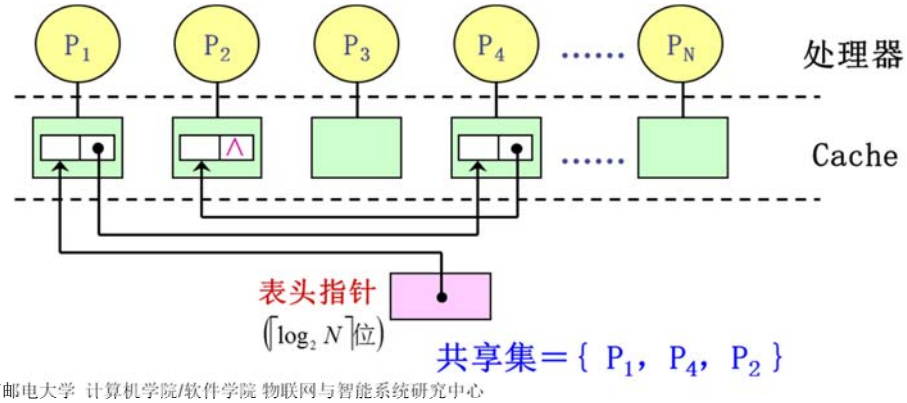
- 提高其可扩放性和减少目录所占用的空间
- 核心思想：采用位数固定的目录项目
  - 限制同一数据块在所有Cache中的副本总数。
  - 例如，限定为常数m。则目录项中用于表示共享集合所需的二进制位数为：$m \times \log_2 N$。
  - 目录所占用的空间与$N \times \lceil \log_2 N \rceil$ 成正比。



共享集＝{P_1，P_3，P_4，P_7 } 有限映像目录（m＝4，N≥8的情况）

Chained directory

- 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，其优点是：既不限制副本的个数，又保持了可扩展性。
- 单链法或双链法



同时多线程

- 线程级并行性 – TLP-Thread Level Parallelism
  - 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，而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

实现多线程有两种主要的方法

- 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。

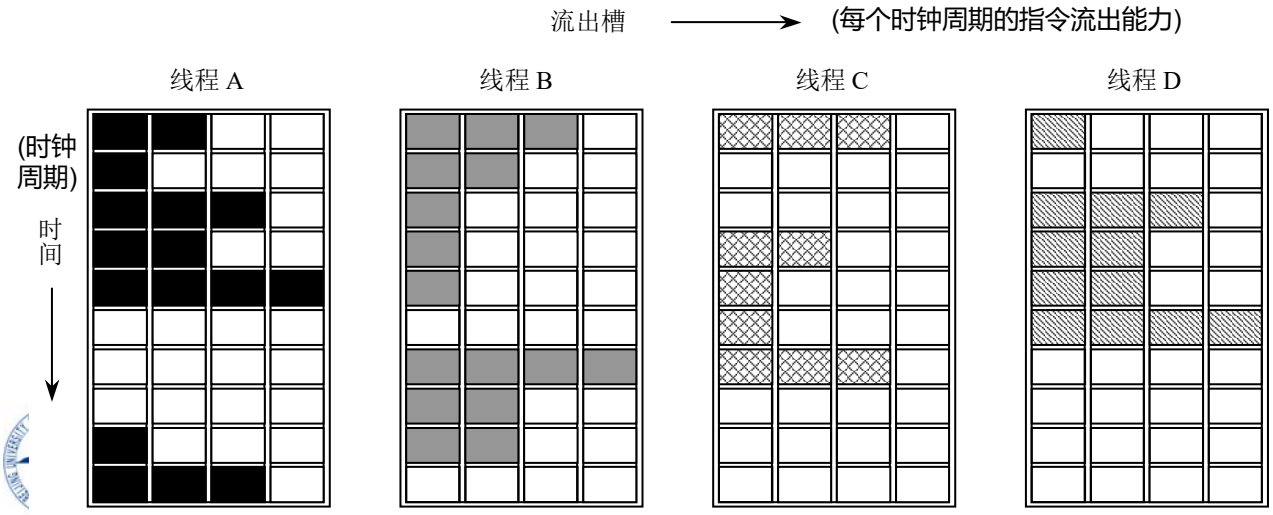
将线程级并行转换为指令级并行

- 同时多线程技术(SMT - Simultaneous Multi Threading)
  - 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。
- 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。

邝坚 北京邮电大学 计算机学院/软件学院 物联网与智能系统研究中心

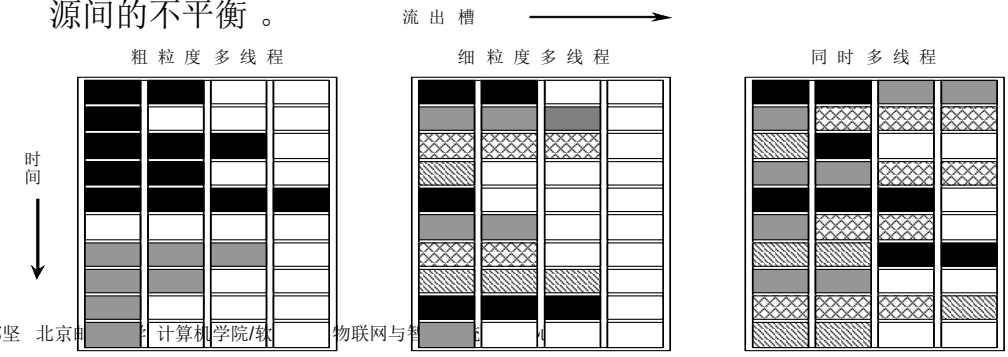
同时多线程技术

- 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。



同时多线程技术

- 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 。



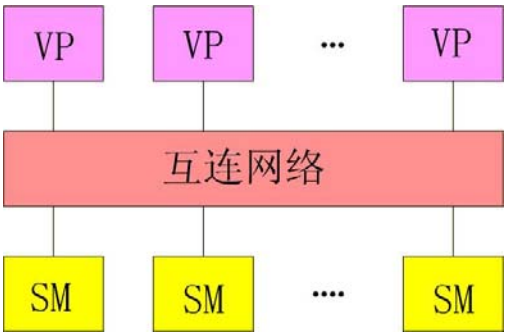
同时多线程技术

- 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。

并行计算机系统结构

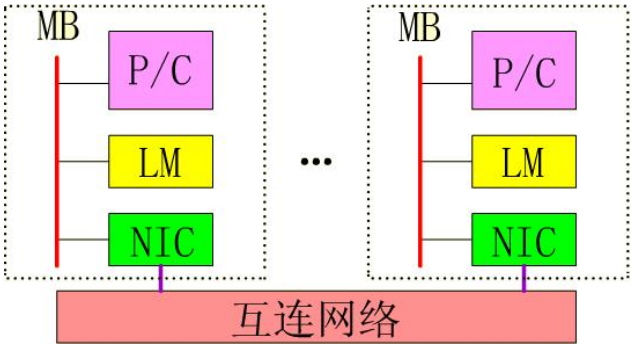
- 目前流行的高性能并行计算机系统结构通常可以分成以下5类：
  - 并行向量处理机（PVP）
  - 对称式共享存储器多处理机（SMP）
  - 分布式共享存储器多处理机（DSM）
  - 大规模并行处理机（MPP）
  - 机群计算机（Cluster）

并行向量处理机(PVP)



- 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。
- Cray C-90和 Cray T-90是这类机器的代表。

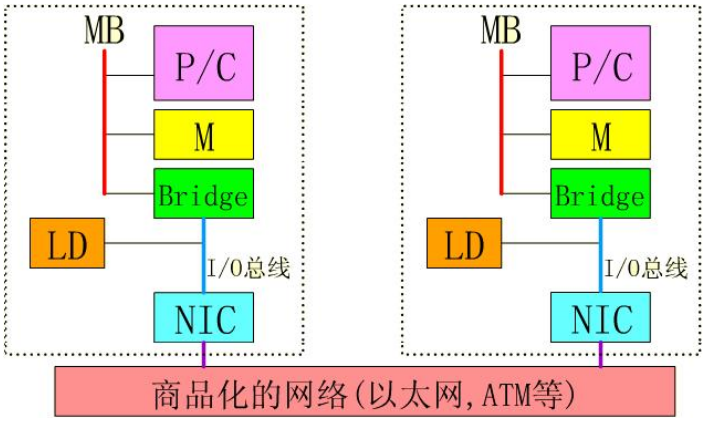
大规模并行处理机(MPP)



Massively Parallel Processing

- 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。
- Intel Paragon和IBM SP2是这类机器的代表。

工作站机群(COW/Cluster)



Cluster Of Workstation

- 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。
- Berkeley NOW和SP2是这类机器的代表。

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