高级体系结构历年试卷 (2001-2017) 总结

杨森

2017 年 9 月 1 日

[left for blank]

基础

名词解释/填空

1. (微处理器技术) 的发展带来了计算机设计的复兴, 既强调 (体系结构) 方面的创新, 也重视技术改进的高效应用.(2006)
2. Throughout this range in price and capabiity, the desktop market tends to be driven to optimize (*price-performance*).(2006)
3. Power Consumption in Computer Systems Power consumption in modern systems is dependent on a variety of factors, including the chip (*clock frequency*), (*efficiency*), the disk drive speed, disk drive utilization, and (*DRAM*).(2013)
4. 服务器的三个关键特征:(可用性),(可扩展性),(吞吐能力).(2006)
5. 一个事件从启动到完成的事件, 称为 (响应时间), 也称为 (执行时间); 仓库级计算机的操作人员可能关心的是 (吞吐量), 也就是给定时间内完成的总工作量.(2006)
6. **计算机体系结构**: 计算机系统结构是指 (传统机器程序员) 看到的 (计算机属性), 即 (概念性结构) 和 (功能特性).(2001、2005、2008、2013)
7. 存储程序计算机 (冯诺依曼结构) 的特点是: 机器以 (运算器) 为中心; 采用存储程序原理; 存储器是按照 (地址) 访问的、线性编址的空间; 控制流由 (指令流) 流产生; 指令有操作码和地址码组成;数据以 (二) 进制编码表示, 采用二进制运算;由 (运算器),(存储器),(输入/输出设备),(控制器) 组成 (2007、2009、2012、2015)
8. 程序的局部性原理是指: 程序总是趋向于使用最近使用过的 (数据) 和 (指令), 程序执行时所访问的存储器地址不是随机分布的, 而是相对簇集, 包括 (时间局部性) 和 (空间局部性):(2008、2012、2015)
   (a) **时间局部性**: 程序即将用到的信息很可能是目前正在用的信息.(2002)
   (b) 空间局部性: 程序即将用到的信息很可能与目前正在使用的信息在空间上邻近.
9. **软件兼容 (Software Compatable)**: 软件兼容是指一台计算机上的程序不加修改就可以搬到另一台计算机上正常运行.(2002)
10. 所谓系列机就是指具有相同的 (体系结构), 但具有不同 (组成) 和 (实现) 的一些列不同型号的机器. 我们把不同厂家生产的具有相同体系结构的计算机称为 (兼容机).(2008、2010、2013)
11. **并行性 (parallelism)**: 并行性是指计算机系统在同一时刻或者同一时间间隔内运行多种运算或者操作, 包括同时性和并发性两种含义, 同时性是指两个或两个以上的时间在同一时刻发生, 并发性是指两个或两个以上的事件在同一时间间隔内发生.(2002)
12. **Amdahl 定律**: 当对一个系统中的某个部件进行改进后, 所能获得的整个系统性能 (加速比), 受限于该部件的 (执行时间) 占总执行时间的 (百分比).(2004、2010、2011、2014、2016)
13. 器件发生故障的概率与 (时间) 的关系可以使用**学习曲线**来说明。(2011)
14. 我们把 DRAM 这种不供电数据丢失的存储器称为 (易失性) 存储器，把磁盘这种不供电数据也能够保存数据的存储器称为 (非易失性) 存储器。(2015)

简答题

1. 简述存储程序计算机的特点:(2002)

   机器以运算器为中心；采用存储程序原理；存储器是按照地址访问的、线性编址的空间；控制流由指令流流产生；指令有操作码和地址码组成；数据以二进制编码表示, 采用二进制运算；由运算器, 存储器, 输入/输出设备, 控制器组成。

指令系统

名词解释/填空

1. 设计指令系统包括 (寻址方式)，(操作类型)，(指令集操作/数据类型)，(指令系统编码) 等方面。(2011、2016)
2. **RISC** 含义是 (精简指令集计算机),**CISC** 含义是 (复杂指令集计算机).(2008、2015)
3. **寄存器-寄存器型 (Load/Store 型)** 指令结构: 操作数都来自通用寄存器组.(2001、2003)
4. 指令系统的基本要求:
   (a) 完整性: 在有限可用的存储空间内, 对于任何可解的问题, 编制计算程序时, 指令系统提供的功能足够使用.
   (b) **规整性**: 指令系统的规整性主要包括对称性和均匀性, 对称性是指所有与指令系统有关的存储单元的使用、操作码的设置等都是对称的；均匀性是指对于各种不同的操作数类型、字长、操作种类和数据存储单元, 指令的设置都要同等对待.(2001、2004)
   (c) **正交性**: 指令中各个不同含义的字段, 如操作类型、数据类型、寻址方式字段等, 在编码时应该互不相关、相互独立.(2002)
   (d) 高效率: 指令的执行速度快、使用频率高.

简答题

1. *CISC* 结构计算机的缺点和 *RISC* 结构计算机的设计原则:(2004)

   CISC 是复杂指令集计算机，缺点主要有以下几点：

   - 各种指令的使用频率相差悬殊，许多指令很少用到。
   - 指令系统庞大，指令条数很多，许多指令功能又很复杂，这使得控制器硬件变得非常复杂。
   - 许多指令由于操作繁杂，其 CPI 值比较大，执行速度慢。
   - 由于指令功能复杂，规整性不好，不利于利用流水线来提高性能。

   RISC 是精简指令集计算机，设计原则主要包括以下几点:

   - 指令条数少、指令功能简单；
   - 采用简单而统一的指令格式，减少寻址方式；
   - 指令的执行在单周期内完成 (采用流水线技术);
   - 采用 load-store 结构，即只有 load 和 store 指令采用访问存储器，其他指令的操作都是在寄存器之间进行的；
   - 大多数指令都采用硬连逻辑实现；
   - 强调优化编译器的作用，为高级语言程序生成优化的代码；
   - 充分利用流水技术来提高性能。

流水线

名词解释/填空

1. 流水线分类:

   (a) 部件级流水线: 把处理机中的部件进行分段, 在把这些分段相互连接而成.

   (b) 处理机级流水线: 又称指令流水线 (Instruction Pipeline), 把指令的执行过程按照流水方式进行处理, 即把一条指令的执行过程分解为若干个子过程.

   (c) 系统级流水线: 又称为宏流水线 (Macro Pipeline), 把多个处理机串行连接起来, 对同一数据流进行处理, 每个处理机完成整个任务中的一部分.

   (d) 单功能流水线 *(Unifunction Pipeline)*: 流水线各段之间连接固定不变, 只能完成单一固定功能.

   (e) **多功能流水线 (Multifunction Pipeline)**: 流水线各段之间可以进行不同的连接, 以实现不同功能.(2004)

   (f) 静态流水线 *(Static Pipeline)*: 同一时间内, 多功能流水线的各段只能按照同一种功能的连接方式工作.

   (g) 动态流水线 *(Dynamic Pipeline)*: 同一时间内, 多功能流水线的各段可以按照不同的方式连接.

   (h) 线性流水线 *(Linear Pipeline)*: 流水线各段串行连接、没有回馈.

   (i) 非线性流水线 *(Nonlinear Pipeline)*: 各段除了有串行的连接外, 还有反馈回路.

   (j) 顺序流水线 *(In-order Pipeline)*: 流水线输出端任务流出顺序与输入端任务流入顺序完全相同.

   (k) 乱序流水线 *(Out-order Pipeline)*: 流水线输出端任务流出顺序与输入端任务流入顺序可以不同.

2. 对于采用指令流水线的处理器, 其 CPI 公式为: CPI 流水线 = 流水线理想 CPI + (结构相关停顿导致的停顿) + (数据相关导致的停顿) + (控制相关导致的停顿).(2005、2007、2013)

3. 流水线的理想 CPI 是衡量流水线 (最高) 性能的一个指标.(2006)

4. 相关分类

   (a) **结构相关**: 当指令在同步重叠执行过程中, 硬件资源满足不了指令重叠执行的要求, 发生资源冲突.(2001、2002、2003)

   (b) **数据相关**: 当一条指令需要用到前面的执行结果, 而这些指令均在流水线中重叠执行是产生数据相关; 对于顺序指令 i 和 j, 指令 i 和 j 存在数据相关是指, 指令 j 的 (操作数寄存器或存储器) 是指令 i 要写入的寄存器或者 (存储单元)。(2006、2010)

   (c) 控制相关: 当流水线遇到分支指令和其他能够改变 PC 值的指令就会发生控制相关.

   (d) **反相关**: 如果指令 j 的 (目标寄存器) 是指令 i 要 (访问) 的 (寄存器) 或 (存储单元).(2011、2014、2016、2017)

   (e) 输出相关: 如果指令 j 和指令 i 所写的名相同, 则称指令 i 和 j 发生了输出相关.

5. **吞吐率 (ThroughPut, TP)**: 单位时间内流水线所完成的任务数量或者输出结果的数量.(2002)
6. **定向 (Forwarding)**: 在发生写后读相关的情况下, 将计算结果从起产生的地方直接从到其他指令需要它的的地方.(2003、2017)
7. **分支延迟槽 (Brancy Delay Slot)**: 存放分支指令的后继指令，无论分支指令成功与否，都会执行分支延迟槽中的指令.(2017)
8. **异常 (Exception)**: 指令正常的执行顺序得到改变.(2017)

向量处理机

名词解释/填空

1. 通过时间: 第一个任务从进入流水线到流出结果的时间段.
2. 排空时间: 最后一个任务从进入流水线到流出结果的时间段.
3. 流水线的**链接 (pipeline chaining)** 是将流水线计算机中多个 (功能部件) 按照指令要求链接在一起, 构成一个 (长流水线), 减少各个功能部件流水线 (加载/建立) 和 (排空) 的时间, 提高流水线执行的效率; 向量流水线的链接是在有 (数据 (写后读)) 相关的 (两) 条指令之间, 将产生数据的指令部件的 (结果) 直接送到使用数据部件的 (向量寄存器/输入).(2007、2009、2010、2011、2013、2014、2016)

指令级并行

名词解释/填空

1. 当指令之间不存在 (相关) 时，它们在流水线中是可以重叠起来并行执行的，这种指令序列中存在的潜在并行性称为 (指令级并行)。(2015)
2. **基本块 (Basic Block)**: 一段连续的代码除了入口和出口, 没有其他的分之指令和转入点, 这称其为基本快.(2003)
3. This potential overlap among instructions is called (*instruction-level parallelism (ILP)*) since the instructions can be evaluated in (*parallel*).(2006)
4. 静态调度 *(Static Scheduling)*: 在编译期间而非程序执行过程中进行代码的调度和优化.
5. **动态调度 (Dynamic Scheduling)**: 在程序的执行过程中, 依靠专门硬件对代码进行调度.(2004)
6. **乱序流出 (Out of order issue)**: 程序中的 (指令) 不是按照排列的顺序流出, 而是当指令需要的资源条件得到满足时即 (流出).(2001、2010)
7. 乱序执行 *(Out of order Execution)*: 指令的执行顺序与程序顺序不同.
8. 乱序完成 *(Out of order completion)*: 指令的完成顺序与程序顺序不同.
9. **分支预测缓冲 (Branch Prediction Buffer, BPB, BHT)**: 仅使用一片存储区域, 记录最近一次或几次的分支特征的历史, 包括两个步骤, 分支预测和预测为修改.(2004)
10. 分支目标缓冲 *(Branch Target Buffer, BTB)*: 将分支成功的分支指令和它的分支目标地址都放到一个缓冲区中保存起来, 缓冲去以分支指令的地址作为标识.
11. 在前瞻执行 (speculation) 中, 指令的流出过程是 (顺序) 的, 但结束过程的确认是 (顺序) 的, 再定序缓冲用于保存执行完毕但尚未顺序确认的指令.(2009)
12. **再定序缓冲 (ReOrder Buffer)**: 暂存指令执行的结果, 使其不直接写回到寄存器或者存储器, 能够在分支错误的情况下恢复现场.(2001)
13. **超标量技术 (Superscalar)**: 在每个时钟周期流出多条指令, 但流出的指令的条数不固定.(2003、2017)
14. **全局代码调度 (Globa code scheduling)**: 把分支之前的指令调度到分支之后, 或者把分支之后的指令调度到分支之前, 称之为 (全局) 代码调度, 其目的实在保持代段 (数据) 相关和 (控制) 相关不变的前提下, 将一段内部包含 (控制相关) 相关的代码段压缩到尽可能最短.(2005、2017)
15. **软流水**:(循环重组技术), 使得 (循环体) 由原来不同循环中指令组成. 是一种用于程序中 (循环结构) 的指令 (重组) 技术, 又称为 (符号化循环展开).(2002、2005、2007、2010)

简答题

1. 多指令流出主要受哪些方面的限制:(2003)

   处理器中指令的流出能力是有限的，主要受以下三个方面的影响:

- 程序所固有的指令集并行性。这一限制是最简单也是最根本的元素，对于流水线处理器，需要有大量可并行执行的操作才能避免流水线出现停顿。
- 硬件实现上的困难。多流出的处理其需要大量的硬件资源，因为每个周不仅要流出多条指令，还要执行它们。随着每个时钟周期流出指令数的增加，所需的硬件成正比例的增长。同时所需的存储器带宽和寄存器带宽也大大增加了。
- 超标量和超长指令字处理器固有的技术限制。超标量处理器无论采用记分牌技术还是 Tomasulo 技术都需要大量的硬件，VLIW 处理器仅需要很少甚至不需要额外的硬件，因为这些工作全部由编译器完成。设计多流出处理器要在各个因素和技术之间进行权衡取舍。

2. 基于硬件前瞻执行的优缺点：

   前瞻执行结合了动态分支预测、前瞻执行后续指令、动态调度的三种思想，能够允许指令乱序执行，但要求按程序顺序确认。

   - 优点：前瞻执行可以有效的应对控制相关，基于硬件的前瞻和动态调度相结合，可以做到系统结构相同但实现不同的机器能够使用相同的编译器。
   - 缺点：所需的硬件太复杂。

存储系统

名词解释/填空

1. 在存储层次中, “Cache-主存” 层次为了弥补主存 (速度) 不足, “主存-辅存” 层次为了弥补主存 (容量) 的不足.(2008、2011、2012、2015、2016)

2. **虚拟 cache**: 虚拟 Cache 是指可以直接用虚拟地址进行访问的 Cache, 其标识存储器中存放的是虚拟地址, 进行地址检测用的也是虚拟地址.(2001、2009)

3. **存储体冲突**:(多体交叉存储器) 中, 两次独立的存储器访问请求在 (一个存储周期) 中访问 (同一个) 存储体, 就造成存储体冲突.(2001、2009、2011、2014、2016)

4. Cache 的三种类型不命中

   (a) **强制性不命中 (Compulsory Miss)**: 当第一次访问一个块时, 该块不在 Cache 中, 需要从下一级存储器中调入 Cache, 这就是强制性不命中.(2002、2004、2010)

   (b) **容量不命中 (Capacity Miss)**: 程序执行时所需的块不能全部调入 Cache, 则当某些快被替换后, 若又重新被访问, 就会发生容量不命中.(2003)

   (c) 冲突不命中 *(Conflict Miss)*: 在组相联或者直接映像 Cache 中, 若太多的块映像到同一组 (块) 中, 则会出现某个块被别的块替换, 然后又被重新访问的情况, 就会发生冲突不命中.

5. 映像规则 (2005)

   (a) **全相联映像 (Fully Associative)**: 把主存中任意一块放置到 Cache 中任意一个位置

   (b) **直接映像 (Direct Mapping)**: 主存中每一块放置到 Cache 中唯一位置

   (c) **组相联映像 (Set Associative)**: 主存中每一块放置到 Cache 中唯一一个组中任意位置

6. 预取方式

   (a) 寄存器预取: 把数据取到寄存器中

   (b) *Cache* 预取: 把数据取到 Cache 中

   (c) 故障性预取: 预取时, 若出现 (虚地址故障) 或 (违反保护权限), 就会发生异常

   (d) **非故障性预取 (非绑定预取,nonbinding)**: 当出现虚地址故障或违反保护权限时, 不发生异常, 而是放弃预取, 转为空操作. 非绑定预取能够返回 (最新数据值), 并且保证对数据实际的存储器访问的是最新的数据项 (2008、2009)

7. TLB 是一个专用的 (高速缓存部件), 用于存放近期经常使用的 (页表项), 其内容是页表部分内容的一个 (副本).(2007、2012)

简答

1. 简述两级 *Cache* 的工作原理:(2001)

   答: 在原有 Cache 和存储器之间增加另一级 Cache, 构成两级 Cache, 这就可以把第一级 Cache 做的足够小, 使其能够和快速 CPU 的时钟周期相匹配; 同时把第二级 Cache 做的足够大, 使其能捕获更多本来需要到主存去的访问, 降低实际不命中开销, 克服 CPU 和主存之间的性能差距, 使存储器和 CPU 性能匹配。

2. 简述 *TLB* 的工作原理 (2001):

   TLB 用于存放近期经常使用的页表项，其内容是页表部分内容的一个副本。进行地址转换时，直接查找 TLB，只有在 TLB 不命中时，才需要访问内存中的页表，也成为块表或地址变换缓冲器，是一种能够实现快速地址变换的技术。

3. *cache*-主存与主存-辅存层次的区别:(2004)

	Cache-主存层次	主存-辅存层次
目的	为了弥补主存速度不足	为了弥补主存容量不足
存储管理的实现	全部由硬件实现	主要由软件实现
访问速度的比值 (第一级比第二级)	几比一	几万比一
典型块大小	几十个到几百个字节	几千个以上字节
CPU 对第二级的访问	可直接访问	均通过第一级
不命中时 CPU 是否切换	不切换	切换到其他进程

4. 减少 *Cache* 失效开销的策略 (2002):

   采用两级 Cache；读不命中优先于写；写缓冲合并；请求字处理技术；非阻塞 Cache

5. 减少 Cache 命中时间的策略:

   容量小且结构简单的 Cache；虚拟 Cache；访问流水化；多体 Cache；路预测；Trace Cache；

6. 降低 Cache 不命中率的策略:

   调节 Cache 块大小，增加 Cache 容量；提高相联度；采用 Victim Cache；采用伪相联 Cache；硬件预取；编译器控制的预取；编译优化；

互连网络

名词解释/填空

1. 在拓扑上, 互连网络为输入和输出两组节点之间提供了一组互连 或映像.
2. **均匀混洗 (shuffle)**: 输入左移一位;**超立方体路由**: 特定为置反;设互联网的输入为 $(x_{k-1},\cdots,x_1,x_0)$, 则均匀混洗输出是 y=$(x_{k-2},\cdots,x_1,x_0,x_{k-1})$, 超立方体输出是 y=$(x_{k-1},\cdots,\bar{x_j},\cdots,x_1,x_0)$。(2010)
3. 静态互连网络: 由点和点直接相连而成
4. 动态互连网络: 由开关通道 实现, 可以动态改变结构
5. 节点度: 与节点相连接的边的数目称为节点度, 这里的边表示链路或通道, 进入节点的通道数为入度, 从节点出来的通道数为出度 (节点度 = 入度 + 出度)(2008).
6. **网络直径**: 网络中任意两个节点间 (最短) 路径长度的 (最大值) 叫做 (网络直径).(2002、2004、2006、2009、2010、2012、2014、2016、2017)
7. **等分宽度**: 将网络切成 (任意相等两半的各种切法中), 沿切口的 (最小通道边数) 称为等分带宽.(2009、2011、2013、2014、2015)
8. 对于一个网络, 如果从任何一个节点看, 拓扑结构都一样, 则称此网络为对称网络.
9. 路由: 在网络通信中对路径的选择与指定.
10. **虚拟自适应**: 将一个 (物理通道) 分成几个 (虚拟的通道), 根据后续各虚拟通道的 (资源或网络) 情况自适应选择后续通道。(2003、2007、2012、2015)
11. **虫孔路由 (Worm hole)**: 把信息包分成 (小片), 片头带 (目的地址), 所有片以不可分离的 (流水方式) 通过片缓冲区进行传输路由.(2001、2004、2007、2012)

简答题

1. 简单比较动态网络中总线、多级网络、交叉开关的特点:(2003、2008、2009)
   - 总线结构: 系统总线在处理机、I/O 子系统、存储模块或辅助存储设备之间提供了一条公用通信通路。主动设备或主设备产生访问存储器的请求, 被动设备或从设备则响应请求。在多个请求情况下, 总线仲裁分配总线的使用权。公用总线是在分时基础上工作的。
   - 多级网络: 多级网络可用于构造大型多处理机系统。每一级都采用了多个开关, 相邻级开关之间都有固定的级间链接。主要有点是采用模块结构, 可扩展性较好, 然而其时延随着网络的级数二上升。
   - 交叉开关: 它把 N 台处理机和 M 个存储器连接起来, 网络中的每个交叉点是一个允许任何一台处理机和任何一个存储器连接的开关。属于无阻塞置换网络。交叉开关的硬件复杂性较高, 但交叉开关的带宽和路由性能最好。

多处理机

名词解释/填空

1. **Home 结点 (宿主结点)**: 是指存放 (需要访问) 的 (存储单元) 及 (相应的目录项) 所在的结点.(2008、2009、2016)

2. 本地结点 *(local node)*: 发出 (访问请求) 的结点.

3. 远程结点 *(remote node)*: 拥有 (被访问存储块副本) 的结点.

4. 拥有者 *(owner)* 是指拥有唯一的 (*Cache* 块副本) 的处理器.(2008)

5. **栅栏同步**: 栅栏强制所有的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.(2003、2007、2013、2016)

6. 同时多线程 (Simultaneous Multi Threading, SMT) 是同时实现 (指令级) 和线程级的并行, 每拍有 (多个指令槽), 可以安排多个线程的 (多条指令) 同时流出。(2005、2009、2011、2014、2015)

7. 基本硬件原语: 原子交换 *(Atomic Exchange)* 将一个存储单元的值和一个寄存器的值进行交换; 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; 读取并加 *1(fetch_and_increment)* 返回存储单元的值并自动增加该值; 指令对 *LL/SC*, 从第二条指令的返回值判断该指令执行是否成功, 这两条指令之间不能插入其他对存储单元进行操作的其他指令.

8. 机器的同步方法:(2010、2012、2013、2014、2017)

   (a) 硬件方法:

   - **硬件排队锁**: 通过硬件向量等方式进行排队和同步控制。
   - **硬件原语**: 引进一种原语 (可以是 Fetch_and_increment) 减少栅栏技术时所需的时间, 从而减小串行形成的瓶颈。

   (b) 软件方法:

   - **延迟等待旋转锁**: 加锁失败时, 延迟的时间指数增大;
   - **软件排队锁**: 用数组将要进行同步的进程排队, 按照排序进行同步操作;
   - **组合树栅栏**: 通过组合树 (Combining Tree) 减少栅栏中进程读取 release 标志形成的冲突。

9. MIMD 计算机分类、存储器系统结构

   (a) 集中式共享存储器结构 *(Centralized Shared-Memory Architecture)*: 处理器较少, 共享一个集中式的物理存储器。因为主存单一, 且对各处理器的关系是对称的, 所以称其为**对称式共享存储器多处理机 (Symmetric shared-memory Multi processor, SMP)**, 因此也称为 **UMA(Uniform Memory Access)** 结构。

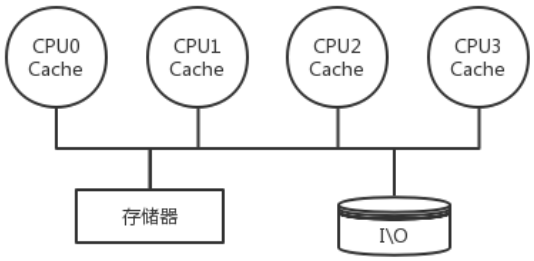
   (b) 分布式存储器多处理机: 存储器在物理上是分布的, 支持的多处理机系统, 具有两种存储器系统结构和处理器通信方式:

- 把物理上分离的所有存储器进行统一的 (共享逻辑空间) 进行编址，任何处理器都可访问任何存储单元，这类系统称为**分布式共享存储器系统 (Distributed Shared-Memory, DSM)**，此处共享是 (地址空间上) 的共享，不具有集中的存储器，也被称为 **NUMA**,，这是因为其 (访存时间) 取决于 (数据) 在存储器中的 (存放位置)。(2011、2013、2014、2015、2017)
- 把每个结点中存储器编址为一个 (独立的地址空间)，不同结点地址空间独立，每个结点中存储器只能有本地处理器进行访问，远程处理器不能对其进行直接访问，这种计算机系统多以 (机群) 形式存在。

简单题

1. 共享主存多处理结构 *(UMA)* 及其特点:(2002)

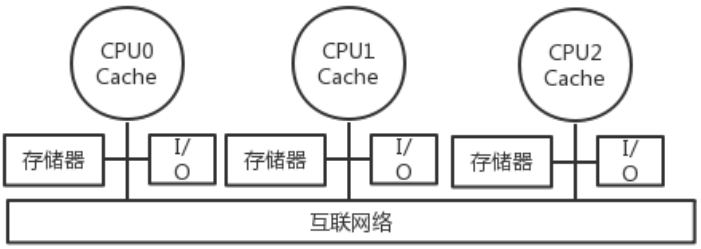
图 1: 共享主存多处理器结构



特点：处理器数目较少，可通过大容量的 Cache 和总线互联使各处理器共享一个单独的物理存储器；只有一个单独的主存，而且这个主存对于各处理的关系是对称的，从各处理器访问它的时间相同；由于增加处理器会使可能的数据通路数量大大增加，从而导致可用带宽减小，因此 UMA 可扩展性差。

2. 分布式存储器结构及其特点：

图 2: 分布式存储器结构



特点：可以支持较大数目的处理器，系统中每个结点包含了处理器、存储器、I/O 以及互联网络的接口。分布式存储器结构需要高带宽的互联网络。分布式存储器结构带来的好处包括，第一如果大多数的访问是针对本节点的局部处理器，则可降低对存储器和互联网络的带宽要求，

第二对局部存储器的访问延迟低。分布式存储器结构最主要的缺点是处理器之间的通信较为复杂。

3. 多处理 Cache 的一致性及其解决方案:

   如果允许共享数据进入 Cache，就可能出现多个处理器的 Cache 中都有同一存储快的副本的情况，当其中某个处理器对其 Cache 中的数据就行修改后，就会使其 Cache 中的数据与其他 Cache 中的数据不一致。这就是多处理机的 Cache 一致性 (Cache coherence)。

   保持一致性要求的两种协议模式：

   - 写作废协议 (Write Invalidate)：在处理器写某个数据项之前保证它对该数据项有唯一的访问权，具体做法是在进行写入之前，把所有其他 Cache 中的副本全部作废；
   - 写更新协议 (Write Update): 在一个处理器写某个数据项时，通过广播使其他 Cache 中所有对应的数据项副本进行更新。

   写更新协议和写作废协议的性能差别来自三个方面:

   - 对同一数据的多个写而中间无读操作的情况，写更新协议需要进行多次写广播操作，而写作废协议只需一次作废操作；
   - 对同一块中多个字进行写，写更新协议对每个字的写均要进行一次广播，而在写作废协议下仅对本快的第一次写时进行作废操作即可。写作废是针对 Cahce 块进行操作的，写更新是针对字 (或字节) 进行操作的；
   - 从一个处理器写到另一个处理器读之间的延迟通常在写更新模式中较低，因为它在写数据是马上更新了相应的其他 Cache 中的内容 (假设读的处理器 Cahce 中有此数据)，而在写作废协议中，需要读一个新的备份。

4. 存储器一致的三个条件:

   - 处理器 P 对 X 单元进行一次写之后又对 X 进行读，读和写之间没有其他处理器对 X 单元进行写，则读的返回值总是写进的值；
   - 一个处理器对 X 单元进行写之后，另一个处理器对 X 单元进行读，读和写之间无其他写，则读 X 单元的返回值应为写进的值；
   - 对同一单元的写是顺序化的，即任意两个处理器对同一单元的两次写，从处理器看来顺序是相同的。

5. 什么是多处理机的相关性和一致性:(2007、2008、2010、2011、2012、2014、2015、2016、2017)

   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，则称该存储系统是一致的，定义包括两个方面：相关性，即返回给读操作的是什么值；一致性，即什么时候才能将已写入的值返回给读操作。相关性和一致性是互补的，相关性定义了对同一个存储器地址进行的读写操作，而一致性定义了访问其他存储器地址上的读写操作。

6. 监听协议的工作原理:(2003、2005、2010、2014)

   每个 Cache 中除了包含物理存储器中块的数据备份之外，也保存着各个块的共享状态信息。Cache 通常连接在共享存储器的总线上，各个 Cache 控制器通过监听总线来判断他们是否有总线上请求的数据块。实现监听协议的关键有三个方面：

   - 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 Cache 响应本地处理器的访问时，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。
- 所有的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，则响应消息，并进行相应的操作。

获取总线控制权的顺序性保证了写操作的串行化。

Cache 发送到总线上的消息事务主要有以下三种：读不命中、写不命中、作废。其中，“写不命中” 和 “读不命中” 表示本地 CPU 对 Cache 进行读访问和写访问不命中，这时都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。对于写直达 Cache，数据最新值由存储器提供，对于写回 Cache，若数据最新值在某个处理器的 Cahce 中，则由该处理 Cache 向请求方提供该快，并停止由读不命中和写不命中引发的对存储器的访问；“作废” 用来通知其他处理器作废相应的副本。

Cache 本来就有的标识用来实现对总线监听，通过把总线上的地址和 Cache 内的标识进行比较，就能找到相应的 Cache 块；通过每个块的有效位可以实现作废机制，当要作废一个块时，只需将其有效位置为无效即可；为了分辨某个数块是否共享，通过给每个 Cache 块增设一个共享为来表示该快是处于共享状态还是专有状态。

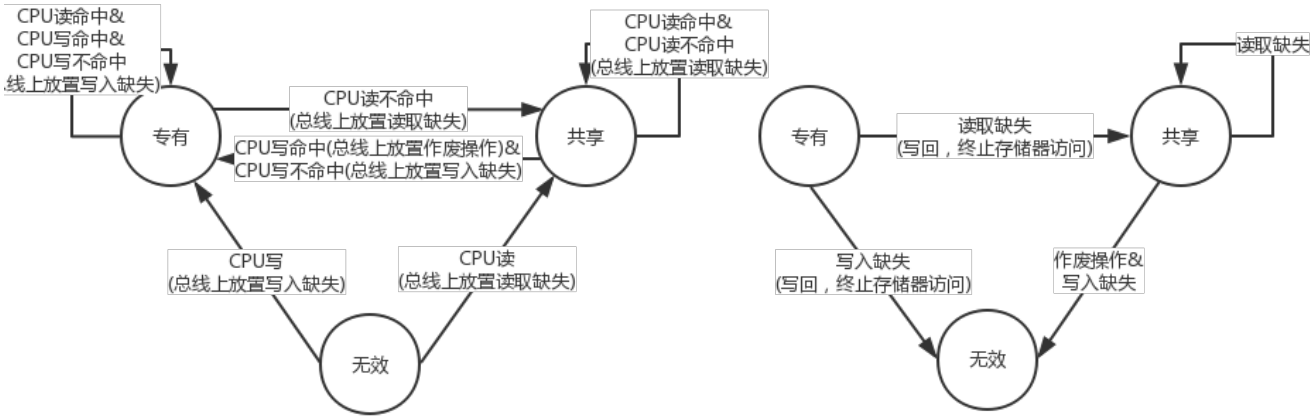
在监听协议中假设操作具有原子性，即操作进行过程中不能被打断。

在一个基于总线的集中共享多处理中，监听协议下 Cahce 块有以下三种状态，每个数据块的状态只能为其中一种：

- 共享: 在一个或多个处理器上具有该快的副本，且主存中的值为最新值；
- 无效: 所有处理器中的 Cache 都没有该快的副本；
- 专有: 仅有一个处理器上由此快的副本，且已对此块进行了写操作，而主存中的数据块仍为旧的。

响应 CPU 请求的 Cache 状态转换如左图所示，处理机响应来自总线请求的状态转换如右图所示：

图 3: 监听协议状态转换图



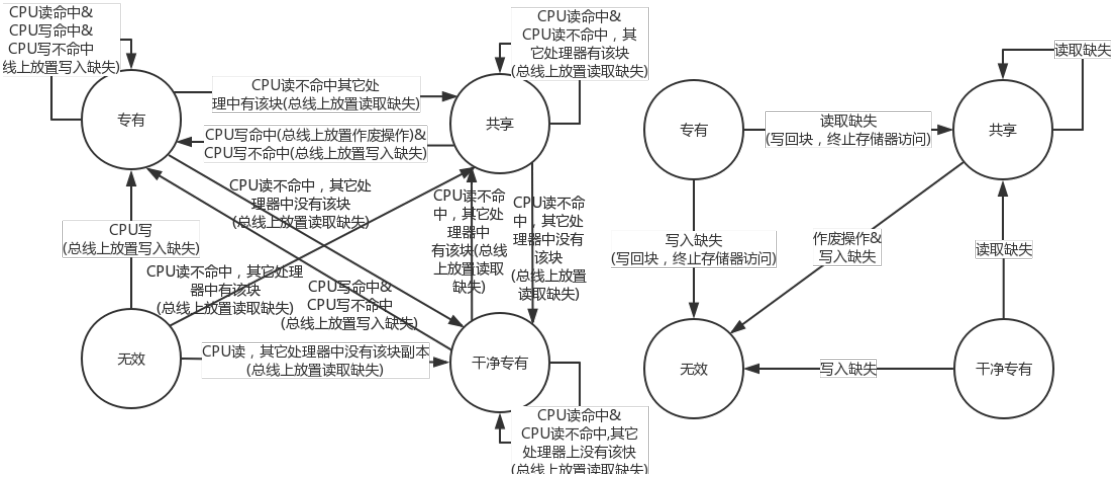
写直达 Cache 与写回 Cache 最大的区别在于，本地处理器不需要读取另一个处理器的 Cache 块。从而在写直达协议中不在提供硬件在读失效或写失效是将被替换的块强制写回内存，也

不再因访问其他 Cache 而中断处理器访问。主存在在 CPU 每次写 Cache 时都会更新，所以在处理器产生读失效后就会直接访问主存。在写直达 Cache 中，共享或专有的 Cache 块都与主存保持一致性。

7. 加入干净专有 (只读) 状态后的监听协议工作原理:(2006、2012、2017)

增加干净专有状态后，当 CPU 读取缺失时，首先判断其他处理器上是否有该块的副本，若有，则将本地 Cache 块状态标记为“共享”，否则标记为“干净专有”；当其它处理器试图读取“干净专有”块时，其状态转换为“共享”；对“干净专有”块写后，其状态转换为“专有”。

响应 CPU 请求的 Cache 状态转换如左图所示，处理机响应来自总线请求的状态转换如右图所示：



8. 目录协议的工作原理:(2007、2011、2015)

物理存储器中共享数据块的状态及相关信息均被保存在一个称为目录的地方。它记录着可以进入 Cache 的每个数据块的访问状态、该快在各个处理器的共享状态以及是否修改过等信息。目录协议通常采用位向量的方法记录那些 Cache 中有副本；目录协议根据位向量中的信息和当前要进行的操作，依次对相应的 Cache 发送控制消息，并完成对目录项信息的修改。

目录协议中，存储块的状态有三种：

- 无效：该块尚未调入 Cache，所有处理器中都没有这个块的副本；
- 共享：一个或多个处理机上有该快的副本，且这些副本与存储器中的该快相同；
- 专有：仅有一个处理机中有该块的副本，且该处理机已对其进行了写操作，所以存储器中该块的数据已经过时。

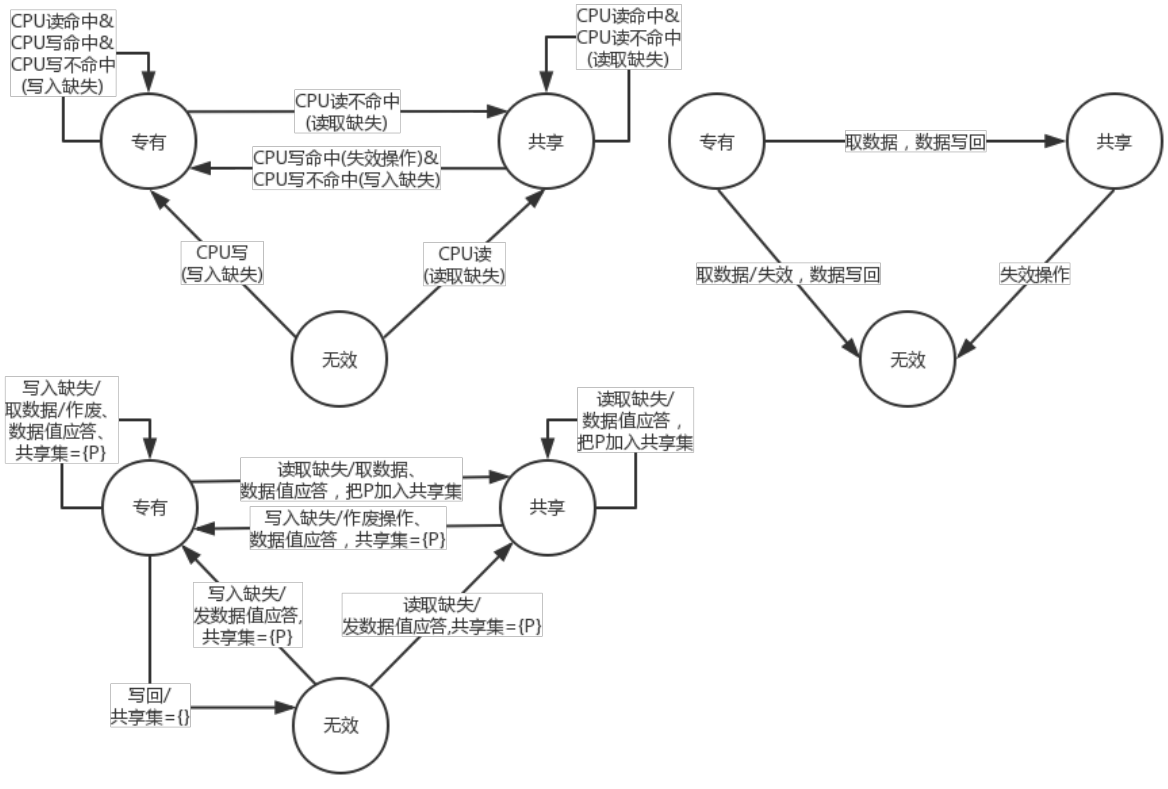
在目录协议中，本地结点是发出请求的结点，主节点是指包含所访问的存储单元及其目录项的结点，远程结点是拥有相应存储快副本的结点。

发往一个目录的消息会产生两种不同类型的操作：更新目录状态和发送消息满足请求服务。结点间传递的消息类型如下表所示：

响应本地 Cache CPU 请求时 Cache 的状态转换图如左图所示，远程结点中 Cache 块响应来自宿主结点的请求的状态转换图如右图所示。下图是 Cache 在各个状态下所接收到的请求和进行的响应操作。

表 1: 目录协议中结点间传递的消息类型

消息类型	来源	目标	消息内容	消息的功能
读失效	本地结点	主结点	P，A	结点 P 读取地址 A 不命中，请求数据并将 P 加入共享集
写失效	本地结点	主结点	P，A	结点 P 写地址 A 不命中，请求数据并使 P 成为专有者
作废	本地结点	主结点	A	请求向所有包含地址 A 块的远程 Cache 发送作废请求
作废	主节点	远程结点	A	作废远程 Cache 中包含地址 A 的数据块
取数据	主节点	远程结点	A	从远程 Cache 中取出包含地址 A 的数据块，并送至主节点，把远程 Cahce 中该块的状态设为共享
取数据并作废	主节点	远程结点	A	从远程 Cache 中取出包含地址 A 的数据块，送至主节点，然后作废远程 Cache 的那个块
数据值应答	主节点	本地结点	D	主节点返回数据值
写回	远程结点	主节点	A，D	写回地址 A 的数据值



9. 一台超结点的分布共享多处理机，结点内部运行监听协议，结点之间运行目录协议，试述其工作原理:(2008、2009、2013、2016)

10. 监听协议与目录协议的对比: 监听协议的优缺点：当 Cache 发生不命中时，在总线上广播相应的消息，容易实现、成本较低，然而当系统规模变大时，大量的总线广播会使得总线变成系统的瓶颈。

11. LL/SC 同步原语的原理：

    LL 指令称为链接载入或锁定载入指令，SC 指令称为条件存储指令。对于链接载入指令所指定的存储器位置，如果其内容在对同一位置执行条件存储之前发生了改变，那么条件存储指令就会执行失败。如果两条指令之间进行了上下文切换 (不能有的处理器的指令在链接载入和条件存储指令之间执行)，那么条件存储也会执行失败。

12. 栅栏同步怎样完成同步过程*?(2011、2015)*；什么是栅栏同步*?(2012、2017);* 在标准的栅栏同步中，设单个处理器的通过时间 *(包括更新计数和释放锁)* 为 C，给出 N 个处理器一起进行一次同步所需要的时间表达式。*(2004、2012、2014、2017)*；在采用 k 元组合树的栅栏同步中，设单个处理器的通过时间 *(包括更新计数和释放锁)* 为 C，求 N 个粗利器一起进行一次同步所需的时间。*(2005)*:

    (a) 栅栏同步过程:

    栅栏同步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。栅栏的典型实现是用两个旋转锁，一个用来保护一个计数器，另一个用来封锁进程直至最后一个进程到达栅栏。栅栏的实现要不停的指定变量，直到它满足规定的条件。下面是一个典型的实现：

```
lock(counterlock);
if(counter==0)release=0;
count=count+1;
unlock(counterlock);
if(count==total){
  count=0;
  release=1;
}
else{
  spin(release);
}
```

    (b) 标准栅栏同步时间

    忽略读写锁的时间。N 个处理器中的每一个都需要 C 个周期来锁住与栅栏相关的计数器并修改它的值,然后释放锁。由于锁只能顺序访问计数器,在同一时间只能有一个处理器修改计数器值，所以共需要 NC 个周期使得所有的处理器都到达栅栏。这是在默认存在排队锁提高栅栏性能的情况下，若不存在排队锁，则 N 个处理器争用时，时间为 NC，N-1 个处理器争用时，时间为 (N-1)C，以此类推，则总的时间为 $NC+(N-1)C+\cdots+C=\frac{NC(N+1)}{2}$

    (c) k 元组合树栅栏同步时间

    k 元组合树是多个请求在局部结合起来形成数的一种分级结构，局部组合的分支数量大大小于总的分支数量，因此组合树降低冲突的原因是将大冲突化解成为并行的多个小冲

突。每个结点组合 K 个进程，提供一个单独的计数器和锁，因而在每个结点有 k 个进程进行竞争，当 K 个进程都到达树中，对应结点则进入父节点，然后递增父节点计数器，当父节点计数到达 K 时，置 release 标志。则所需的时间为 $\lceil log_k N \rceil * kC$。

13. 标准栅栏存在的问题，及其改进措施：

    假设所有进程在离开栅栏时，其中有一个进程还没有离开栅栏，即停留在旋转等待操作上，这是如果有新的进程有又到达了栅栏，而上一次循环的进程最后那个还没来得及离开栅栏，那么这个块进程会将 release 重新置为 0，从而将慢进程捆在栅栏上，这样所有的进程都会处于无限等待状态，因为进程的总数达不到 total。

    改进的办法是当进程离开栅栏是进行计数，在上次栅栏使用的所有进程离开前，不允许任何进程重用并初始化栅栏，这样会明显增加栅栏的延迟和竞争。另一种方法是 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。下面是一个 sense_reversing 的实现：

```
local_sense=!local_sense;
lock(counterlock);
counter++;
unlock(counterlock);
if(count==total){
  count=0;
  release=local_sense;
}
else{
  spin(release==local_sense);
}
```

14. 比较同时多线程 *(Simultaneous Multithreading)*、粗粒度多线程和细粒度多线程及其特点:(2006、2010、2013)

    - 细粒度多线程能够在每条指令之间进行线程切换，从而使多个线程交替执行，通常以时间片循环的方式实施线程交替，在循环过程中跳过阻塞的进程。
      - 优点：能够隐藏由于长时间或短时间阻塞引起的吞吐率的缺失。
      - 缺点：降低了每个线程的执行速度，这是因为及时没有任何阻塞的线程也不能连续执行，而且会因为其他线程指令的插入执行而被延迟。
    - 粗粒度多线程的切换只发生时间较长的阻塞出现时候，通过线程间的切换部分隐藏了代价较高、时延较长的阻塞带来的吞吐率的损失。
      - 优点：降低了线程的切换次数，不会减慢每个独立线程的执行。
      - 缺点：减少吞吐率损失的能力有限，特别是对于较短的阻塞来说更是如此。
    - 同时多线程在多流出、动态调度的处理器上同时开发线程级并行性和指令级并行性，所有的发射槽在一个时钟周期内被多个线程共享，线程级并行和指令级并行被同时开发。
      - 优点：提高吞吐率，且不损失单个线程的性能。
      - 缺点：实际中的一些因素，如线程活跃的个数、缓冲大小限制、可并行指令及从多个线程中取指的能力，仍将限制流出槽的利用率。

15. 同时多线程的并行工作原理及其在体系结构实现上的基础:(2007)

- 工作原理:

  同时多线程是一种在多流出、动态调度的处理器上同时开发线程级并行性和指令级并行性的技术。同时多线程通过使用多发射和动态调度处理器同时实现指令级和线程级的并行，通过寄存器重命名和动态调度，多个线程的指令可以被同时发射，而不考虑指令间的相关性；相关由动态调度负责处理。在同时多线程中，所有的发射槽在一个时钟周期内被多个线程共享，线程级并行和指令级并行被同时开发。

- 结构基础:

  使用动态调度技术的处理器已经具备了开发线程级并行所需的硬件设置。具体来说，动态调度超标量处理器有大量的虚拟寄存器，可以用来保存每个独立线程的寄存器状态。由于寄存器重命名机制提供了唯一的寄存器标识符，多个线程的指令可以在数据路径上混合执行，而不会导致个线程间源操作数和目的操作数的混乱。

  通过乱序执行机制，可以很好的利用硬件功能，提高并行性。利用重排序缓存，将来自独立线程的指令以独立的方式提交。在以上描述的硬件基础上，只需通过在支持单线程的处理器上，为每个线程实现设置重命名表，保留各自的 PC 值，提供多个线程的指令结果提交的能力来实现。

- 具体实现:
  - 设置多个独立的重命名表，以保存每个独立线程的寄存器状态。
  - 让取值部件和指令 Cache 并发预取多个线程指令流，并分别设置各自的 PC；
  - 指令完成时，处理器为不同的线程提供指令结果提交。

16. 松弛一致性模型、特点及其硬件支持 (2003、2006、2007、2009、2010、2012、2014、2015、2017):
松弛一致性模型的关键特点是允许乱序执行读取和写入操作，但是用同步操作来实施排序。同步程序的表现就像处理器具备顺序连贯性一样。松弛一致性在保证程序正确性前提下增加了指令执行的并行，所以采用松弛一致性模型的机器可以提高性能。

根据消除的读取顺序的内容，可以将松弛一致性模型划分为四类:

- 完全存储排序模型，消除 $W \leftarrow R$ 顺序。在硬件支持上为写缓冲的读旁路等。维护写的次序，这种模型采用写缓存，并提供读的旁路机制，能够允许处理机在其写的操作被所有别的处理机看到之前就继续进行读。
- 部分存储排序模型，消除了 $W \leftarrow W$ 顺序。在硬件支持上为写的流水线或其他写并行等。允许非冲突写隐含地乱序进行。实现上，可以使写流水化或重叠，而不是强制一个操作必须在另一个之前结束，对同步操作扔需将写操作挂起，因为它引起写保护。
- 弱排序模型，进一步消除了 $R \leftarrow W$，$R \leftarrow R$ 顺序，在硬件支持为不封锁读。
- 释放一致性模型，进一步消除了 $W \leftarrow S_A$，$R \leftarrow S_A$，$S_R \leftarrow W$，$S_R \leftarrow R$，在硬件支持上为不封锁读、旁路、无序写等。这种模型区分同步操作中的访问一个共享变量的获取操作 S_A 和释放操作 S_R。

17. 大规模机器的同步有哪些软件和硬件方法:(2011、2016)

(a) 软件方法

- 旋转锁：是指处理器环绕一个锁不停地旋转而请求获得锁，当锁的占用时间很少以及加锁过程延迟很低时可以采用旋转锁

- 软件排队锁：可以排队记录等待的进程，当锁释放时发送出一个已经确定的等待进程。软件实现主要是采用记录等待进程的排队数组进行排队
- 组合树栅栏：是多个请求在局部结合起来形成树的一种分级结构，局部组合的分支数量大大小于总的分支数量，因此组合树栅栏降低冲突的原因是将大冲突化解称为多个并行的小冲突

(b) 硬件方法

- 硬件排队锁：可以排队记录等待的进程，当锁释放时发送出一个已经确定的等待进程。硬件实现一般是在基于目录的机器上，通过硬件向量等方式来进行排队和同步控制。
- 硬件原语：用硬件实现，能够以原子方式自动读出和修改存储单元

18. 时延隐藏技术:(2001、2004、2007、2013)

MPP 系统采用了包括数据预取、相关性 Cache、松弛一致性、多现场这几种时延隐藏技术。

- 数据预取：在数据使用之前就将其取到近处；
- 相关性 Cache：减少对共享数据访问的竞争及时延；
- 松弛一致性：在保证程序正确性前提下增加指令执行的并行；
- 多现场：在产生时延时进行现场切换转而执行其他程序。

计算题

流水线

1. 2001. 三.2: 多功能动态流水线、时空图、吞吐率、效率、加速比

   令

   $$x_1 + y_1 = d_1, x_2 + y_2 = d_2, x_3 + y_3 = d_3$$
   $$x_4 + y_4 = d_4, x_5 + y_5 = d_5, x_6 + y_6 = d_6$$
   $$d_1 \times d_1 = c_1, d_3 \times d_4 = c_2, d_5 \times d_6 = c_3$$
   $$c_1 \times c_2 = e_1, e_1 \times c_3 = e_2$$

   采用最快的处理方式，流水线的时空图为：

5				d1	d2	d3	d4	d5	c1	d6	c2		c3		e1				e2
4			d1	d2	d3	d4	d5		d6										
3		d1	d2	d3	d4	d5		d6											
2							c1	c1	c2	c2	c3	c3	e1	e1			e2	e2	
1	d1	d2	d3	d4	d5	c1	d6	c2		c3		e1				e2			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

   吞吐率 $= \frac{\text{任务数}}{\text{时钟周期数}} = \frac{11}{19}$
   效率 $= \frac{44}{95}$
   加速比 $= \frac{\text{不采用流水线所用的时间}}{\text{采用流水线所用的时间}} = \frac{4\times11}{19} = \frac{44}{19}$

存储系统

1. 2001. 三.4

   当块的大小分别为 32,64,128 时，1KB、16KB 和 64KB 的 Cache 失效率最低。失效率最低时，平均访存时间为：

   $$5ns + 100ns \times 13.34\% = 18.34ns$$
   $$5ns + 100ns \times 7.00\% = 12ns$$
   $$5ns + 100ns \times 1.02\% = 6.02ns$$

2. 2002. 三.1

   局部失效率 $= \frac{\text{该级 Cache 的不命中次数}}{\text{到达该级 Cache 的访存次数}}$
   全局失效率 $= \frac{Cache}{CPU}$
   对于第一级 Cache，局部失效率 = 全局失效率 $= \frac{40}{1000} = 4\%$
   对于第二级 Cache，局部失效率 $= \frac{20}{40} = 50\%$,全局失效率 $= \frac{20}{1000} = 2\%$

向量处理机

1. 2002. 三.2

(a) 每次循环需要执行 9 条指令，因此需要的时间为 $9 \times 5 \times n + 2 \times 5 = 45n + 10$ 个时钟周期。

(b) 调度后的指令为 总的时钟周期数为 $5 + (9n + 2 - 1) = 9n + 6$，吞吐量为 $\frac{9n+2}{9n+6}$, 时空图略

```
        LW      R1,a
        ADDI    R8,R30,#8n
Loop:   LW      R2,0(R30)
        MULT    R2,R1,R2
        ADDI    R30,R30,#8
        SUB     R20,R8,R30
        LW      R4,0(R31)
        ADD     R4,R2,R4
        SW      R4,0(R31)
        BNZ     R20,Loop
        ADDI    R31,R31,#8
```

(c)

指令级并行

1. 2003. 三.3

(a) 不进行代码顺序调整的情况下，代码执行的时钟周期如下 因此不进行代码顺序调整情况

```
loop:   ld      f5,0(r1)    1
        ld      f6,0(r2)    2   load 指令延迟 2 个时钟周期
        stall               3
        stall               4
        multd   f7,f5,f6    5   浮点乘法执行 4 个时钟周期
        stall               6
        stall               7
        stall               8
        addd    f4,f4,f7    9
        addi    r1,r1,#8    10
        addi    r2,r2,#8    11
        subi    r3,r3,#1    12  与分支指令相关，延迟一个周期
        stall               13
        bnez    r3, loop    14
```

下，执行一次迭代的时间为 14 个时钟周期

(b) 调整后的代码执行顺序如下：

loop:	ld	f5,0(r1)	1	
	ld	f6,0(r2)	2	load 指令延迟 2 个时钟周期
	addi	r1,r1,#8	3	
	addi	r2,r2,#8	4	
	multd	f7,f5,f6	5	浮点乘法执行 4 个时钟周期
	stall		6	
	stall		7	
	subi	r3,r3,#1	8	与分支指令相关，延迟一个周期
	addd	f4,f4,f7	9	
	bnez	r3, loop	10	

多处理机

其他

1. 超线性加速比:(2002. 四.2) 一些科学应用程序通常会在小幅增加处理器数目时 (2 或 4, 增加到 8 或 16)，实现超线性加速比。这些结果的出现通常是因为一些关键性的数据结构无法放入拥有 2 个或 4 个处理器的多处理中的聚合缓存，但却可以放入拥有 8 个或 16 个处理器的多处理的聚合缓存中。

2. 2003. 三.1

   (a) 机器 A 的平均每条指令的执行时间为 $1.8 \times 20ns = 36ns$，机器 B 的平均每条指令的执行时间为 $1.2 \times 25ns = 30ns$

   (b) $\frac{\frac{1}{36\times 20}}{\frac{1}{30\times 25}} = \frac{25}{24}$, 因此机器 A 的性价比更高。