高级体系结构总结

杨森

2017 年 10 月 5 日

第一章　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时间, 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; 指令由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使用的信息.(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事件在同一时刻发生, 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. 大概率事件优先原则：对于大概率事件，赋予它优先的处理权和资源使用权。

13. **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)

14. 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)

15. 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，把磁盘这种不供电也能够保存数据的存储器称为 (非易失性) 存储器。(2015)

16. 模拟 *(Simulation)*: 用软件方法在一台现有计算机上实现另一台计算机的指令系统。

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；指令由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。

第二章　指令系统

名词解释/填空

1. 设计指令系统包括 (寻址方式设计),(指令集功能设计),(操作数表示和数据类型),(指令系统编码设计)。(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)；把指令集设计成只包含那些频率高的少量指令，并提供一些必要的指令以支持 (操作系统) 和 (高级语言)，按照这个原则设计的计算机称为精简指令集计算机。(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 指令才能访问存储器，其它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。

第三章　流水线

名词解释/填空

1. 流水线分类:

   (a) 部件级流水线 *(Arithmetic Pipeline)*: 把处理机中的部件进行分段, 再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. 冲突/冒险 *(Hazard)*: 由于指令之间存在依赖关系，使得指令流中的下一条指令不能在指定的时钟周期开始执行。

5. 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 (真相关)**: 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单元) 是指令 i 要写入的寄存器或者 (存储单元)。或者指令 j 与 k 数据相关，指令 k 与指令 i 数据相关，则指令 j 与指令 i 数据相关。(2006、2010)

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

(d) 名相关：指令使用的寄存器或存储器称为名，如果两条指令使用相同的名，但是他们之间没有数据流，则称之为名相关。

i. 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。

ii. **反相关**: 如果指令 j 的 (目标寄存器或存储单元) 是指令 i 要 (访问) 的 (寄存器) 或 (存储单元)，则称指令 i 和指令 j 发生了反相关。(2011、2014、2016、2017)

6. 解决流水线瓶颈的方法有 (细分瓶颈段) 和 (瓶颈段重复设置)。

7. **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)

8. **定向 (Forwarding, Bypassing)**: 在某条指令产生一个计算结果之前，其它指令并不真正需要该计算结果，如果能够将该计算结果从其产生的地方直接送到其它指令需要它的地方，就可以避免因数据相关引起的停顿，这就是定向技术。(2003、2017)

9. **分支延迟槽 (Branch 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)

10. 分支延迟 *(Branch Delay)*：由分支指令引起的延迟。

11. **异常 (Exception)**: 异常也称为中断或错误，主要分为两类，一类会导致程序终止运行，另一类则要求在程序处理完异常后继续“透明”地执行原来程序。(2017)

12. 精确异常 *(precise exception)*、非精确异常: 不精确异常是指，当指令 i 导致发生异常时，处理机的现场与严格按程序顺序执行时指令 i 的现场不同。如果发生异常时，处理机的现场与严格按照程序顺序执行时指令 i 是现场相同，就称为精确异常；如果流水线能够准确的判断出异常是由流水线的哪一条指令引发的，并可以正确的保留住这一条指令的现场，则称之为精确异常处理，否则就是非精确异常处理。

13. 流水线互锁 *(Pipeline Interlock)*：用于检测发现数据冲突，并是流水线停顿，直至冲突消失。

14. 分支取消机制 *(Canceling, Nullifying)*：分支指令隐含了预测的分支执行方向。当分支的实际执行方向和事先预测的一样时，执行分支延迟槽中的指令，否则就将该指令转化成空操作。

简答题

1. 降低流水线分支损失的方法：

   冻结 (freze) 或排空 (flush) 流水线，一旦在 ID 段检测到分支指令，就暂停后续指令的执行；预测分支失败；预测分支成功；延迟分支，从逻辑上延长分支指令的执行时间。

第四章　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 (输入).(2007、2009、2010、2011、2013、2014、2016)
4. 分段开采：当向量的长度大于向量寄存器的长度时，必须把长向量分成长度固定的段，然后循环分段处理，每一次循环只处理一个向量段。这种技术称为分段开采技术。

   分段开始的时间计算公式：$T_{all} = \lceil \frac{n}{MUL} \rceil \times (T_{start} + T_{loop}) + mn$ (MUL 表示向量寄存器长度，m 表示编队数，n 表示向量长度，T_{start} = 通过时间 -1,T_{loop} 表示编队执行带来的开销)。

第五章　指令级并行

名词解释/填空

1. 数据级并行 *(Data-level Paralism)*: 在数据并行中，不同机器有着整个模型的完全拷贝；每个机器只获得整个数据的不同部分，计算的结果通过某些方法结合起来。

2. 任务级并行 *(Task-level Paralism)*: 每一个线程执行一个分配到的任务，而这些线程则被分配到该并行计算体系的各个计算节点中去。

3. 循环级并行 *(Loop-level Paralism)*: 循环结构中不同迭代之间的并行性。

4. 线程级并行 *(Thread Level Paralism, TLP)*: 在一种耦合硬件模型中开发数据级并行和任务级并行，这种模型允许线程之间进行交互。

5. 指令级并行 *(Instruction Level Paralism, ILP)*: 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的并行性称为 (指令级并行)。(2015)

6. 循环展开 *(Loop Unrolling)*: 展开循环体若干次，将循环级并行转化为指令级并行的技术。

7. **基本块 (Basic Block)**: 一段连续的代码除了入口和出口, 没有其它的分支指令和转入点, 这称其为基本块.(2003)

8. 指令调度: 通过改变指令在程序中的位置，将相关指令之间的距离加大到不小于指令执行延迟的时钟周期数，这样就可以将相关指令转化为无关指令，指令调度受限于指令固有的指令级并行和功能部件的延迟。

9. This potential overlap among instructions is called (*instruction-level parallelism (ILP)*) since the instructions can be evaluated in (*parallel*)。(2006)

10. 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)。

11. **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)

12. **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)

13. 乱序执行 *(Out of order Execution)*: 指令的执行顺序与程序顺序不同。

14. 乱序完成 *(Out of order completion)*: 指令的完成顺序与程序顺序不同。

15. **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, 分支预测和预测位修改.(2004)

16. 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, 缓冲区以分支指令的地址作为标识.

17. 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)

18. **再定序缓冲 (ReOrder Buffer)**: 指令前瞻执行时在确认阶段需要一套额外的硬件缓冲来保存那些执行完毕但未经确认的指令及其结果，这种硬件的缓冲称为再定序缓冲，ROB 的每个项包括指令类型、目的地址、值域和就绪字段。(2001)

19. **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，可以通过编译器静态调度，也可以通过记分牌或 Tomasulo 进行动态调度.(2003、2017)

20. *VLIW*: 是一种多指令流出技术，采用多个独立的功能部件，将多条指令的操作组装成固定格式的指令包，形成一条非常长的指令。

21. 超流水: 处理器的每个功能部件进一步流水化，被分解成多个段，使得一个功能部件在同一周期能够处理多条指令。

22. **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, 其目的是在保持代段 (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) 的代码段压缩到尽可能最短。(2005、2017)

23. **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)

24. 保留站 *(Reserve Station)*: 保留站中保存已经流出并等待相应功能部件执行的指令，其内容包括操作码、操作数以及用于检测和解决冲突的信息。

25. 路径调度: 通过增加执行频率低的分支开销来简化代码调度。

26. 路径压缩 *(Trace Compaction)*: 将路径中的操作尽可能提前，并将所有操作尽可能压缩成少量长指令，并尽量进行调度。

简答题

1. 基于硬件前瞻执行的优缺点:

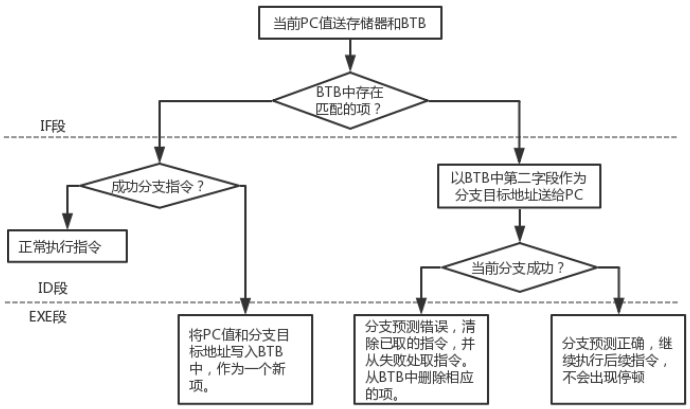
   One of the significant advantages of speculation is its ability to uncover events that would otherwise stall the pipeline early, such as cache misses. This potential advantage, however, comes with a significant potential disadvantage. Speculation is not free. It takes time and energy, and the recovery of incorrect speculation further reduces performance. In addition, to support the higher instruction execution rate needed to benefit from speculation, the processor must have additional resources, which take silicon area and power. Finally, if speculation causes an exceptional event to occur, such as a cache or translation lookaside buffer (TLB) miss, the potential for significant performance loss increases, if that event would not have occurred without speculation.

前瞻执行结合了动态分支预测、控制结果尚未确定时前瞻执行后续指令、跨越基本快的动态调度的三种思想，能很好地解决控制相关的问题。前瞻执行允许指令乱序执行，但要求按程序顺序确认。前瞻执行能够实现精确异常，例如 Cache 不命中，当某条指令引起异常时，等待该指令到达 ROB 的头部再对该指令引起的异常进行处理，同时清除所有正在执行的指令。前瞻执行的最大缺点是所需硬件太复杂。除此之外，当前瞻执行预测错误进行恢复时会降低性能；如果前瞻执行导致异常事件，例如 Cache 不命中或 TLB 不命中，也会导致性能的下降。

2. 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：(2003)

   - 程序所固有的指令级并行性。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。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。

3. 五级流水线中分支目标缓冲的过程：(2017)



4. 记分牌算法和 Tomasulo 算法的基本思想：

   (a) 记分牌：记分牌技术的目标是在资源充足时，尽可能早地执行没有数据阻塞的指令，达到每个时钟周期执行一条指令。如果某条指令被暂停，而后面的指令与流水线中正在执行的或被暂停的指令不相关，那么后面的指令可以继续流出并执行。记分牌电路负责记录资源的使用，相关检测，以及控制指令的流出与执行。

   (b) Tomasulo：只要操作数有效，就将其取到保留站，避免指令流出时才到寄存器中取数据，这就使得即将执行的指令从相应的保留站中取得操作数，而不是从寄存器中。指令的执行结果也是直接送到等待数据的其它保留站中。

   两者的区别在于 Tomasulo 算法将冲突检测和指令执行区分开，指令计算的结果通过相关通路直接从功能部件送入对应的保留站中而不一定是写寄存器。与之相比，记分牌集中控制冲突的检测和指令的执行，将结果首先写入及寄存器。

5. 开发指令级并行的技术和方法：

   开发指令级并行的技术和方法包括指令静态和动态调度、解决控制相关技术和多指令流出技术。指令的静态调度是由编译器完成的，包括循环级并行的处理、寄存器换名和指令调度等。动态调度包括记分牌技术和 Tomasulo 算法。解决控制相关的技术包括分支预测缓冲技术、分支目标缓冲技术和前瞻执行技术。多流出技术实现了在每个周期流出多条指令，主要包括超标量技术和超长指令字技术。

第六章　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 (同一个存储周期) 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, 则当某些块被替换后, 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, 若多个的块映像到同一组 (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返回的是最新的数据项 (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. 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

2. 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；

3. 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；

4. 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，这样就可以把第一级 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。

5. 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，也称为快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。

6. *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	切换到其它进程

第七章　互连网络

名词解释/填空

1. 互连网络：将对称系统或分布式系统中的节点连接起来所构成的网络，这些节点可能是处理器、存储器模块或其它设备，它们通过互连网络进行信息交换。

2. 片上网络: 将“报文交换”的思想引入到片上互连结构中，构成了“片上网络”, 它具有更高的可扩展性和可重用性。

3. 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.

4. **均匀混洗 (shuffle)**: 输入左移一位;**超立方体路由**: 特定位置反;设互连网络的输入为 $(x_{k-1},\cdots,x_1,x_0)$, 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)

5. 静态互连网络: 由点和点直接相连而成

6. 动态互连网络: 由开关通道 实现, 可以动态改变结构

7. 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).

8. **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)

9. **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)

10. 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)。

11. 路由: 在网络通信中对路径的选择与指定.

12. **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)

13. **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

14. 通讯延迟：发送开销 + 跨越时间 + 传输时间 + 接收时间。

15. 超结点: 每个结点内可能还包含较小数目的处理器，这些处理器之间互连形成簇，这样形成的结点叫做超结点。

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。在多个请求的情况下，总线通过仲裁分配使用权。公用总线是在分时基础上工作的。总线结构最简单，但争用最严重，可用带宽较窄。
   - 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连接。主要优点是采用模块结构，可扩展性较好，然而其时延随着网络的级数增加而上升。
   - 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，适用于规模较小的网络。

第八章　多处理机

名词解释/填空

1. **Home 节点 (宿主节点)**: 是指存放 (需要访问) 的 (存储单元) 及 (相应目录项) 的节点.(2008、2009、2016)

2. 本地节点 *(local node)*: 发出 (访问请求) 的节点.

3. 远程节点 *(remote node)*: 拥有 (被访问存储块副本) 的节点.

4. 拥有者 *(owner)* 是指唯一拥有 (*Cache* 块副本) 的处理器.(2008)

5. **栅栏同步 (Barrier)**: 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. 旋转锁 *(spin locks)*: 处理器环绕一个锁不停地旋转而请求获得该锁。

7. 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)

8. 同步硬件原语: 用于构造同步操作的基本原语。

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，通过总线互连共享一个集中式的物理存储器。因为主存单一，且这个主存对各处理器的关系是对称的，从各处理器访问它的时间相同，所以称其为**对称式共享存储器多处理机 (Symmetric shared-memory Multi processor, SMP)**，也称为 **UMA(Uniform Memory Access)** 结构。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。(2002)

   (b) 分布式存储器多处理机: 存储器在物理上是分布的，可以支持较大数目的处理器，系统中每个节点包含了处理器、存储器、I/O 以及互连网络的接口。分布式存储器结构需要高带宽的互连网络。如果大多数访问是针对本节点的局部存储器，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。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作为统一的 (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)

- 把每个节点中存储器编址为一个 (独立的地址空间)，不同节点地址空间独立，每个节点中存储器只能由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. 什么是多处理机的相关性和一致性:(2007、2008、2010、2011、2012、2014、2015、2016、2017)

   如果每次读取某一数据项都会返回该数据项的最新写入值，则这个存储器系统就是相关的 (coherent)。相关性 (coherence) 是指读操作可能返回什么值。一致性 (consistence) 是指什么时候读操作才能得到新写入的值。

2. 监听协议的工作原理:(2003、2005、2010、2014)

   Cache 中除了包含数据块的副本以外，也保存着各个块的共享状态信息。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，检测总线上的地址在其 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。

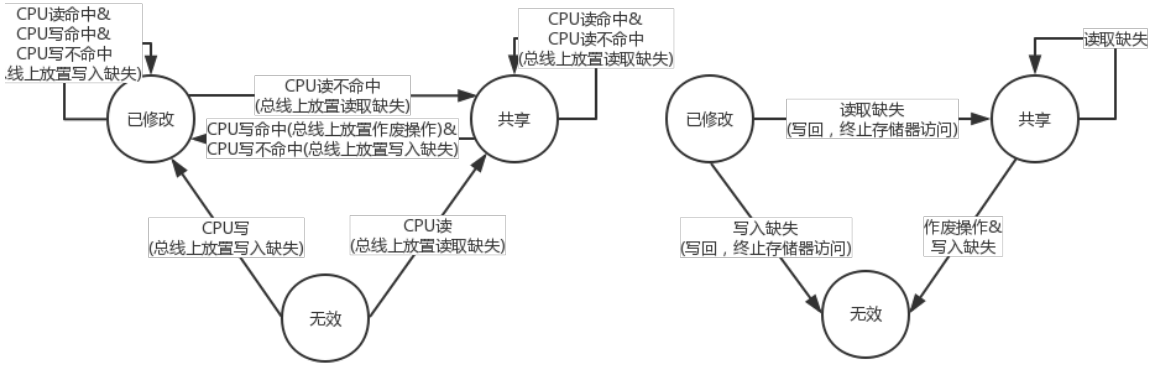
   Cache 发送到总线上的消息事务主要有以下三种：读不命中、写不命中、作废。其中，“写不命中” 和 “读不命中” 表示本地 CPU 对 Cache 进行读访问和写访问不命中，这时都需要通过总线找到相应数据块的最新副本，然后调入本地 Cache。对于写直达 Cache，数据最新值由存储器提供，对于写回 Cache，若数据最新值在某个处理器的 Cahce 中，则由该 Cache 向请求方提供数据块，并停止由读不命中和写不命中引发的对存储器的访问；“作废” 用来通知其它处理器作废相应的副本。

   在一个基于总线的集中共享多处理机系统中，监听协议下 Cache 块有以下三种状态：

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

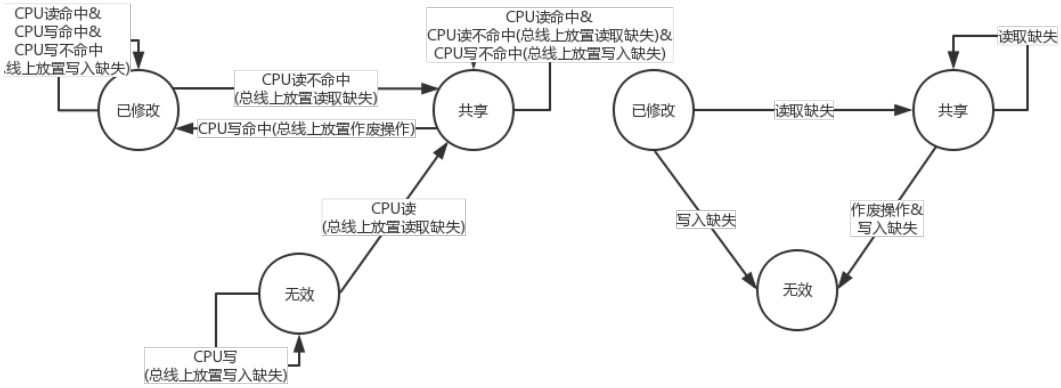
   Cache 本来就有的标识用来实现对总线的监听，将总线上的地址和 Cache 内的标识进行比较，就能找到相应的 Cache 块；通过每个块的有效位可以实现作废机制，当要作废一个块时，只需将其有效位置为无效即可；为了分辨某个数块是否共享，通过给每个 Cache 块增设一个共享位来表示该块是处于共享状态还是已修改状态。

图 8.1: 写回 Cache 下响应本地 CPU 请求和响应总线消息的 Cache 状态转换图



3. 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脏 Cache 块。在写直达 Cache 中，共享或已修改的 Cache 块都与主存保持一致性，当发生读失效或写失效时，不再将被替换的块强制写回内存。CPU 每次写 Cache 时都会更新主存，所以在处理器产生读失效后就会直接访问主存。

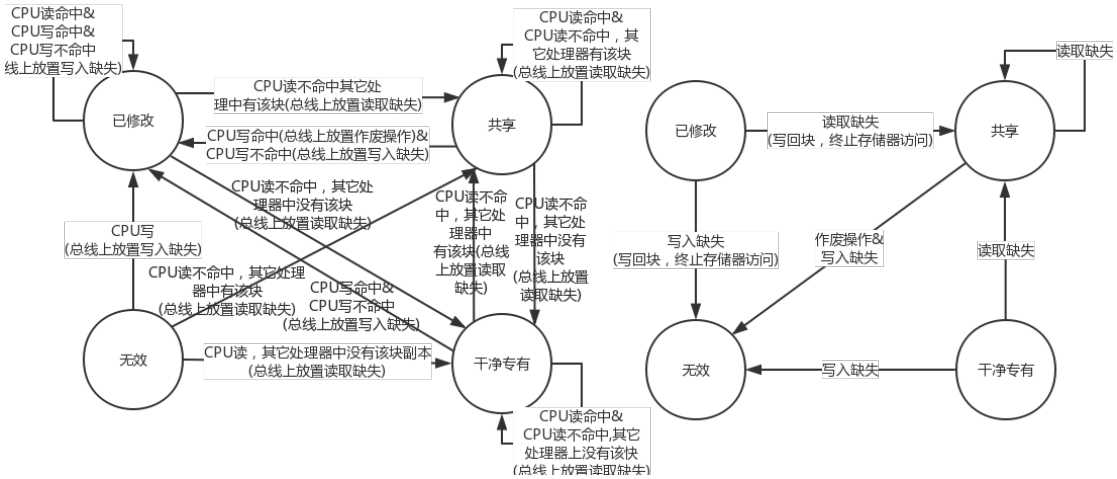
图 8.2: 写直达 Cache 下响应本地 CPU 请求和响应总线消息的 Cache 状态转换图



4. 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，首先判断其它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”；对“干净专有”块写后，其状态转换为“专有”，处于“干净专有”的 Cache 块响应总线上的写入缺失转换为“无效”状态，响应读取缺失转换为“共享”状态。

图 8.3: 增加干净专有状态后，响应本地 CPU 请求和响应总线消息的 Cache 状态转换图



5. 目录协议的工作原理:(2007、2011、2015)

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。存储器的每一个数据块在目录存储器中对应有一项，目录项用于记录该块的状态以及哪些 Cache 中有副本等消息。对于任何一个数据块都可以根据其地址块速找到相关的信息，这使得目录协议避免了广播操作。目录协议常采用位向量记录哪些 Cache 中有副本，位向量中的每一位对应于一个处理器。位向量指定的处理机的集合称为共享集。

目录协议中，存储块的状态有三种：未缓存、共享和独占:

(a) 未缓存状态表示数据块尚未被调入 Cache。

(b) 共享状态表示该块在一个或多个处理机上有该块的副本，且这些副本与存储器中的块相同。

(c) 独占状态表示仅有一个处理机有该块的副本，且该处理机已对其进行了写操作，所以其内容是最新的，而存储器中该块的数据已经过时。

为了提高实现效率，在每个 Cache 中还跟踪记录了每个 Cache 块的状态。在目录协议中有三种节点，分别是:

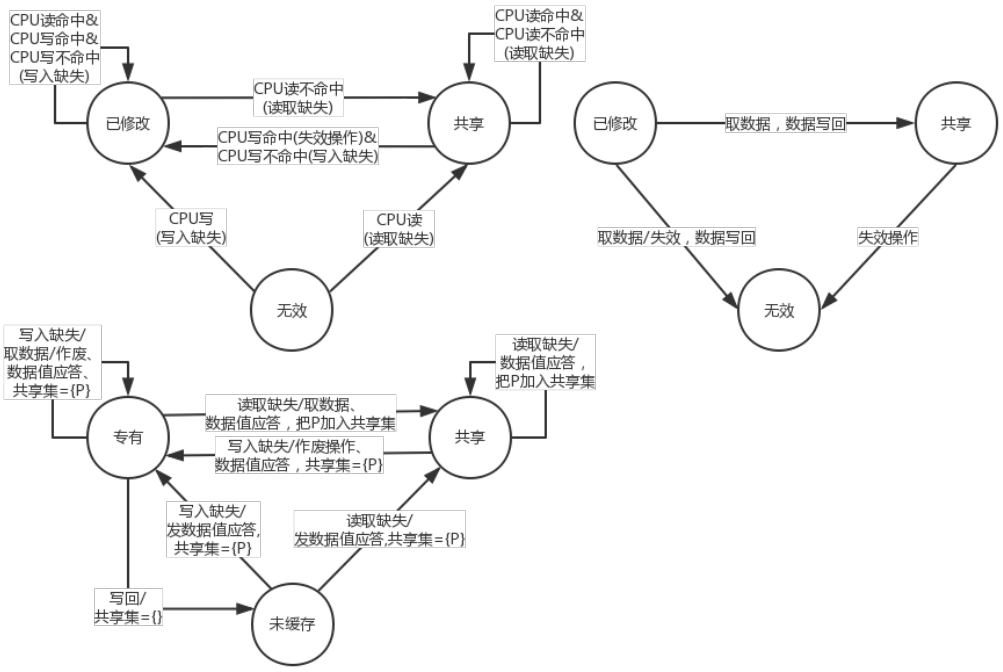
(a) 本地节点，表示发出访问请求的节点。

(b) 宿主节点，表示包含要访问的存储单元及其目录项的节点。

(c) 远程节点，表示拥有相应存储块副本的节点。

目录协议采用点到点的通信。本地节点把请求发送给宿主节点的目录，再由目录控制器有选择地向远程节点发出相应的消息，使远程节点进行相应的操作，并进行目录中状态信息的更新。

图 8.4: 响应本地 CPU 请求的 Cache 状态转换图，远程节点响应宿主节点消息的 Cache 状态转换图，目录接收消息的状态转换图



6. 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)

7. 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。

   顺序连贯性模型保持了 $W \to R, W \to W, R \to R, R \to W$ 四种读写顺序，根据松弛的读取顺序的内容，可以将松弛一致性模型划分为四类:

   - 完全存储排序模型，松弛了 $W \to R$ 顺序。这种模型维护写的次序，采用写缓存，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。在硬件支持上为写缓冲的读旁路等。
   - 部分存储排序模型，松弛了 $W \to W$ 顺序。这种模型允许非冲突写隐含地乱序进行，对同步操作扔需将写操作挂起，因为它引起写保护。在硬件支持上为写的流水化或其它写并行等。
   - 弱排序模型，松弛了 $R \to W$，$R \to R$ 顺序，在硬件支持为不封锁读。
   - 释放一致性模型，松弛了 $W \to S_A$，$R \to S_A$，$S_R \to W$，$S_R \to R$ 顺序，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。在硬件支持上为不封锁读、旁路、无序写等。

8. 大规模机器的同步有哪些软件和硬件方法:(2010、2011、2012、2013、2014、2016、2017)

   (a) 软件方法

- 延迟等待旋转锁：加锁失败时，推延进程的等待时间。一般是在失败时，延迟时间指数增大。
- 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。软件实现用数组将要进行同步的进程排队，按排序进行同步操作。
- 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远小于总的分支数量，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，能够以原子方式读出和修改存储单元。引进一种原语减少栅栏计数时所需的时间，从而减小串行形成的瓶颈。

9. 时延隐藏技术:(2001、2004、2007、2013) 时延隐藏：将通信和计算或多次通信重叠。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它程序。

10. 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 个处理器器一起进行一次同步所需的时间。*(2005)*；采用排队锁和 *Fetch-and-Increment* 实现栅栏同步和旋转锁实现栅栏同步的性能比较:

(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，一个用来保护计数器，该计数器记录已到达该栅栏的进程数，另一个用来封锁进程直至最后一个进程到达栅栏。栅栏的实现要不停地检测指定变量，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。考虑最坏情况，所有 N 个处理器都要对计数器加锁并修改它的值。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的数据，所以总共要花费 NC 个时钟周期使得所有的处理器都到达栅栏。

(c) k 元组合树栅栏同步时间

树中每个节点组合 k 个处理器，提供一个单独的计数器和锁，因而在每个节点有 k 个进程进行竞争。每个节点需要 kC 的时间通过。

树中一共有 $L = \log_k N$ 层，共有节点 $\sum_{i=1}^{L} \frac{N}{k^i}$ 个。因此，最坏情况下所有的节点要串行处理，n 个处理器共需要 $kC \sum_{i=1}^{L} \frac{N}{k^i}$ 的时间完成同步。最快的情况是同一层中的节点并行处理，需要 Lkc 的时间。

(d) 不同栅栏同步实现的性能比较

- 排队锁实现栅栏同步：对 n 个处理器，每个处理器都初始加锁产生 1 个总线事务，其中一个成功获得锁并在使用后释放锁，第一个处理器将有 n+1 个总线事务。每一个后续处理器需要 2 个总线事务：一个获得锁，另一个释放锁。因此总线事务为:$(n+1)+2(n-1)=3n-1$。
- Fetch-and-Increment 实现栅栏同步：对于 n 个处理器，需要 n 此 Fetch-and-Increment 操作，访问 count 变量的 n 次 Cache 失效和释放时 n 次 Cache 失效，总共需要 3n 个总线事务。
- 标准栅栏：i 个处理器竞争锁的时候，包括访问该锁的 i 个 LL 操作，试图锁住该锁的 i 个 SC 操作，以及 1 个释放锁的存操作。因此一共需要总线事务 $\sum_{i=1}^{n}(2i+1) = n^2+2n$ 个总线事务。

11. 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，在循环过程中跳过停顿的进程。
  - 优点：能够隐藏由于长时间或短时间停顿引起的吞吐率的损失。
  - 缺点：降低了每个线程的执行速度，这是因为即使没有任何停顿的线程也不能连续执行，而且会因为其它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停顿出现的时候，通过线程间的切换部分隐藏了代价较高、时延较长的停顿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，特别是对于较短的停顿来说更是如此。
- 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，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。
  - 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，如活跃线程的个数、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。

12. 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。通过动态调度和寄存器重命名机制，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而不考虑指令间的相关性，相关性由动态调度负责处理。

- 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，而不会导致线程间源操作数和目的操作数的混乱。

- 具体实现: 只要在支持单线程的处理器上实现以下几点，就可以实现同时多线程
  - 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确认。

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，这时如果有新的进程又到达了栅栏，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. 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。

15. 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 中都有同一存储块的副本的情况，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 中的数据与其它 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，把所有其它 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，通过广播使其它 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块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，因为它在写数据时马上更新了其它 Cache 中相应的内容 (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。

16. 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，读和写之间没有其它处理器对 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，读和写之间无其它写，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。