高级体系结构知识点

杨森

2017 年 8 月 14 日

基础

名词解释

1. **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)
2. 程序的局部性原理: 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:
   (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。
3. **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)
4. 兼容机: 由不同公司厂家生产的具有相同系统结构的计算机。
5. **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)
6. **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的提高，受限于该部件的执行时间占总执行时间的百分比。(2004)

指令系统

名词解释

1. **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)
2. 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。

流水线

名词解释

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. **结构相关**: 在流水线处理机中，如果某种指令组合因为资源冲突而不能正常执行，则称该处理机有结构相关。(2001、2002、2003)

3. **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)

4. **定向**: 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)

向量处理机

指令级并行

名词解释

1. **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)
2. 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。
3. 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)
4. **乱序流出 (Out of order issue)**: 指令的流出顺序与程序顺序不同。(2001)
5. 乱序执行 *(Out of order Execution)*: 指令的执行顺序与程序顺序不同。
6. 乱序完成 *(Out of order completion)*: 指令的完成顺序与程序顺序不同。
7. **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)
8. 分支目标缓冲 *(Branch Target Buffer, BTB)*:
9. **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)
10. **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)

存储系统

名词解释

1. **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)
2. **存储体冲突**: 两个访问请求要求访问同一个存储体。(2001)
3. 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)
   (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。

互连网络

名词解释

1. 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)。
2. **网络直径**: 网络中任意两个节点间最短路径长度的最大值。(2002)
3. 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。
4. 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。
5. 路由: 在网络通信中对路径的选择与指定。
6. **虫孔路由 (Worm hole)**：把信息包切割成“片”，使信息包中的各片的传送按照流水方式进行。(2001)

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. 静态互连网络: 由点和点直接相连而成
3. 动态互连网络: 由开关通道 实现，可以动态改变结构

多处理机

名词解释

1. **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)

同步

1. 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。

   前三种硬件原语都要执行两次访存操作。LL/SC 可以用于实现其他同步原语例: 用 LL/SC 实现原子交换

$$
\begin{aligned}
try: & \ OR \quad R3, R4, R0 \\
& \ LL \quad R2, 0(R1) \\
& \ SC \quad R3, 0(R1) \\
& \ BNQZ \quad R3, try \\
& \ MOV \quad R4, R2
\end{aligned} \tag{1}
$$

2. *旋转锁 (Spin Locks)*: 处理器不停地请求获得使用权的锁 (锁占用时间少，且加锁过程延迟很低时可采用旋转锁)

3. 使用 Cache 一致性实现锁的好处:1. 可使“环绕”进程 (对锁不停的进行测试和请求占用的循环过程) 只对本地 Cache 中的锁 (副本) 进行操作；2. 可利用所访问的局部性，即处理器最近使用过的锁不久又会使用。

   栅栏的实现需要两个旋转锁，一个用来记录到达栅栏的进程数，另一个用来封锁进程直至最后一个进程到达栅栏。栅栏的一个缺陷: 假设有一个进程还没有离开栅栏，即停留在旋转锁等待操作上。如果另外一个比较快的进程又到达了栅栏，而上一次循环的进程中最后那个还没来得及离开栅栏。那么这个“快”进程就会吧 release 置为 0，从而把上次循环的“慢”进程捆到这个栅栏上，这样所有的进程在这个栅栏上的又一次使用都会处于无线等待状态，因为已经到达栅栏的进程数目永远不会等于 total。