1. **简述两级Cache的工作原理（2001）**

两级Cache是一种降低失效开销的技术。通过在原有Cache和存储器之间增加另一级Cache，构成两级cache，其中第一级cache容量小，但速度与CPU的时钟周期相匹配；第二级Cache足够大，能够捕获更多本来要到主存去的访问，从而能够降低实际失效开销。

访存时先访问一级Cache，若一级Cache命中，则直接获取数据；若一级Cache没命中，则访问二级Cache。二级Cache没有命中才访问存储器。

1. **简述TLB工作原理（2001）**

TLB又称为快表，是一个专用的高速缓冲器，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。这样，进行地址变换时，一般直接查TLB就可以了，只有偶尔在TLB不命中时，才需要去访问内存的页表。TLB也利用了局部性原理，如果访存具有局部性，则这些防村中的地址变换也具有局部性，即所有的页表项是相对聚簇的。TLB也常称为“快表”或地址变换缓冲器。

TLB中的项与Cache中的类似，也是由两部分构成：标识和数据。标志中存放的是虚地址的一部分，而数据部分中则存放物理页帧号，有效位，存储保护信息以及其他一些辅助信息。为了使TLB中的内容与页表保持一直，当修改页表中的某一项时，操作系统就必须奥正TLB有该页表的副本。

1. **什么是I/O和Cache数据一致性问题？如何解决这种数据一致性问题？（2001）**

数据不一致问题有两个方面：

1. 存储器中可能不是CPU产生的最新数据，I/O系统从存储器取出来使用的是陈旧数据。
2. I/O与存储器交换数据滞后，在Cache中被CPU使用的可能就是陈旧数据。

解决措施：

1. 写直达Cache可以保证存储器和Cache有相同的数据。
2. 写回Cache则需要操作系统帮助进行数据检查。
3. 根据I/O使用的存储器地址来清除Cache相应块，确保I/O使用的数据不再Cache中。
4. 地址检查过程也可以使用硬件完成。
5. **简述****存储器程序计算机（冯·诺依曼结构）的特点（2002）**

机器以运算器为中心；采用存储程序原理；存储器是按地址访问的线性编址的空间；控制流由指令流产生；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。

1. **目前实现盘阵列的方式有哪三种？（2002）**

软件方式：即阵列管理软件由主机来实现。优点是成本低；缺点是过多地占用主机时间并且带宽指标上不去。

阵列卡方式：即把RAID管理软件固化在I/O控制卡上，从而可以不占用主机时间，一般用于工作站和PC机。

子系统方式：是一种基于通用接口总线的开放式平台，可用于各种主机平台和网络系统。

1. **简述减少Cache失效开销的策略。（2002）**
2. 读失效优于写：发生读失效时将脏块送至缓冲器，处理读失效再将脏块写入存储器。
3. 子块放置技术：把一个Cache块划分为若干小块各做标记，子块的失效开销小于整体开销。
4. 请求字处理技术：当CPU需要的数据到达后不等整个块到达Cache就把数据发送给CPU。
5. 非阻塞Cache技术：发生miss时不停顿，继续取指执行。
6. 采用两级Cache。
7. **画出共享主存多个处理器系统的概念结构，并说明其特点。（2002）**

处理器数目较小，通过大容量的Cache和总线互联使各处理器共享一个单独集中式存储器。

存储器

I/O

1. **VLIW上性能优于向量处理器的实例。（2002）**

考虑向量处理器RAR的指令相关，即所谓源寄存器冲突，即向量寄存器不能够同时支持

两个向量寄存器读，而VLIW可以。

ADD.D F1 F2 F3

ADD.D F4 F2 F3

ADD.D F5 F2 F3

ADD.D F6 F2 F3

1. **比较动态网络中总线、多级网络、交叉开关的特点。（2003，2008，2009）**

各自特点：

1. 总线结构：通过共享总线把各处理器、共享存储器连接起来，在多个请求情况下，总线仲裁分配总线的使用权，总线互联最简单，但争用最严重。
2. 多级网络：每一级用了多个开关，相邻级开关之间都有固定的级间连接。
3. 交叉开关网络：它把N台处理机和N个存储器连接起来，网络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。

优缺点比较：

1. 总线的造价最低，其缺点是争用严重，每台处理器可用的带宽较窄，且故障率较高。
2. 交叉开关的硬件复杂性以n2倍上升，所以其造价最为昂贵。但是交叉开关的带宽和路由性能最好。如果网络规模较小，它是一种理想的选择。
3. 多级网络则是两种极端的折衷。它的优点在于采用模块化结构，因此可扩展性较好。但其时延随网络级数而上升。另外由于增加了连线和开关复杂性，价格也是一种限制因素。
4. **请给出三种降低Cache命中时间的技术手段。（2003）**

1）采用容量小而结构简单的Cache。

2）采用虚拟Cache，直接使用虚地址进行Cache索引。

3）写操作流水化：对比标识与写入数据流水化，第一次的写操作和第二次的对比标识就可以并行。

1. **给出四种松弛一致性模型的特点及所需要的硬件支持。（2003，2006，2009，2014）**

**松弛一致性模型为什么能提高性能？部分排序和弱排序两种模型的特点及其实现上所需要的硬件支持措施。（2007，2010，2012，2015）**

**采用松弛（relaxed）一致性模型的机器提高性能的原理是什么？实现上有哪些主要的硬件支持措施？（2011）**

答：松弛一致性允许读写操作的乱序执行，但需要同步操作保证排序原则，使得一个同步程序的表现和顺序模型一致。在保证程序正确的前提下，增加指令执行的并行性并减少时延。所以采用松弛一致性模型的机器可以提高性能。

根据消除读写顺序的内容可以划分为四类：

1. 完全存储排序（TSO）：消除了W->R，在硬件支持上为写缓冲的读写旁路等。维护写的次序，这种模型采用写缓存，并提供读的旁路机制，从而允许处理机在写的操作数被所有的别的处理机看到之前就继续进行读。
2. 部分存储排序（PSO）:消除了W->W，在硬件支持上为写的流水线或其他并行等。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操作之前结束，对同步操作仍需将写操作挂起，因为它引起写保护。
3. 弱排序模型（WO）：进一步消除了R->R,R->W，在硬件支持为不封锁读。
4. 释放一致性模型：进一步消除了W->SA，R->SA，SR->R，SR->W，在硬件上支持为不封锁读、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SA和将对象释放允许别的处理机获取访问权的释放操作SR。
5. **多指令流出主要受那些方面的限制？（2003）**

1）程序内在的指令级并行性。

2）硬件实现的限制。

3）超标量和超长指令字处理器固有的技术限制。

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

CISC结构的缺点：

1. 各种指令的使用频率相差悬殊。
2. 指令系统的复杂性带来了计算机体系结构的复杂性，增加了研制时间和成本。
3. 指令系统的复杂性给VLSI设计增加了很大负担，不利于单片集成。
4. 许多复杂指令需要很复杂的操作，因而运行速度慢。
5. 由于各条指令的功能不均衡性，不利于采用先进的计算机体系结构技术来提高系统的性能。

RSIC结构计算机的设计原则：

1. 选取使用频率高最高的指令，并补充一些最有用的指令。
2. 每条指令的功能应尽可能的简单，并在一个机器周期内完成。
3. 所有指令长度均形同。
4. 只用load和store操作指令访问存储器，其他指令操作均在寄存器之间进行。
5. 以简单有效的方式支持高级语言。
6. **给出在多处理机系统中进行时延隐藏的主要技术途径及其工作原理。（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、多现场这几种技术。

1. 数据预取：在数据使用之前就利用存储器的时机将其取到近处。
2. 相关性Cache：减少对共享数据访问的竞争及时延。
3. 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。
4. 多现场：在产生时延时进行现场切换转而执行其他程序。
5. **简述“Cache-主存”层次和“主存-辅存”层次的区别。（2004）**

|  |  |  |
| --- | --- | --- |
| 比较项目 | Cache-主存 | 主存-辅存 |
| 目的 | 为了弥补主存速度上的不足 | 为了弥补主存容量的不足 |
| 存储管理实现 | 主要由专用硬件实现 | 主要有软件实现 |
| 访问速度的比值  （第一级比第二级） | 几比一 | 几万比一 |
| 典型的块（页）大小 | 几十个字节 | 几百到几千个字节 |
| CPU对第二级的访问方式 | 直接访问 | 均通过第一级 |
| 失效时CPU是否切换 | 不切换 | 切换到其他进程 |

1. **简述使用物理地址进行DMA所存在的问题，及其解决办法。（2004）**

存在两个问题：

1. 对于超过一页的数据，由于缓冲区使用的页面在物理存储器中不一定是连续的，传输会发生问题。
2. DMA正在存储器和帧缓冲器之间传输数据时，操作系统从存储器中移除一些页（或重新分配），DMA将会在存储器中错误的页面上传输数据。

解决问题的方法：

采用虚拟DMA技术，允许DMA设备直接只用虚拟地址，在DMA期间由硬件将虚拟地址映射到物理地址。I/O使用的缓冲区在虚拟存储器中要求是连续的，物理页面可以分散在物理存储器中。如果使用虚拟DMA的进程在内存中被移动，操作系统能够及时地修改相应的DMA地址表，或者在DMA期间锁定存储器中相应的页面。

1. **栅栏（Barrier）同步怎样完成同步过程？（2011，2015）**

**什么是栅栏（Barrier）同步？**2012**在标准的栅栏（barrier）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）**

**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个处理器一起进行一次同步所需要的时间。（2005）**

（栅栏同步过程）

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，从而实现同步。栅栏的典型实现是要用两个旋转锁，一个是用来记录到达栅栏的进程数，另一个是用来封锁栅栏，直到最后一个进程到达栅栏。栅栏的实现要不停地指定变量，直到它满足规定的条件。

（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。考虑最坏的情况，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算器的数据，所以共需要花NC个时钟周期使得所有的处理器都到达数据栅栏。

（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树的一种分级结构，局部组合的分支数量大大小于总分支数量，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。变量k表示扇入数目。树中每个节点组合k个进程，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，当第k个进程到达树中对应节点时则进行父节点，然后递增父节点的计数器，当父节点计数器到达k时，置release标志。则所需时间为：logkN\*kC。

1. **在基于总线的集中共享多处理机系统中，Cache相关性监听协议有三个状态：无效、共享和专有。给出在写直达Cache条件相关性监听协议和工作原理及状态变迁图。它与写回式Cache条件下有什么主要区别。（2005）**

**1）监听协议的工作原理详细版**

监听协议即每个Cache除了包含物理存储器中块的数据副本之外，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，每个Cache控制器通过监听总线来判断它们是否有总线请求的数据块。

监听协议关键是对其共享数据写的处理。当某个处理机要写数据时，必须先获得总线的控制权，然后将要作废的数据块地址放在总线上。其他处理机一直监听总线，检测该地址是否存在于他们的Cache中。若存在，则作废相应的数据块。获取总线控制权的顺序性保证了写的顺序性。因为两个处理机同时写一个单元时，其中一个处理机必然先获得总线控制权，之后它使另一个处理机上对应的拷贝作废，从而保证了写的严格顺序性。

当写Cache未命中时，除了将其他处理机上对应的Cache数据块作废外，还要从存储器取出该数据块。（这是写直达Cache情况下的，如果是写回Cache，则需要读取其他处理器脏Cache块）

监听过程的实现可以利用Cache中块的标志位。每个块的有效位使作废机制的实现较为容易。由写作废或其他事件引起的失效处理很简单，只需将改为设置为无效即可。而对于写，我们希望知道是否别的Cache中也有此数据的共享拷贝。因为如果没有别的Cache拷贝，而无需将写地址放在写回Cache的总线上，这样可降低所用的时间和所需的带宽。

为了分辨某个数据块是否共享，需要给每个Cache块加一个特殊的状态位来说明它是否为共享。当对共享块进行写时，本Cache将写作废的请求放在总线上，Cache块状态由共享变为非共享或者专有。如果另一个处理器要求访问该块，则状态会再转化为共享。

在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，例如将写失效的检测、申请总线和接收响应作为一个单独的原子操作。

**2）写直达Cache的状态转换图**



**3）写回式Cache的状态转换图**



**4）写直达Cache与写回式Cache条件下有什么主要区别**

写直达Cache与写回式Cache最大的区别在于，本地处理器不需要读取另一个处理器脏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时将被替换的块的强制写回主存，也不再访问其他Cache拷贝而中断处理器访问。主存在CPU每次写Cache块时候都会更新，所以在处理器产生读失效后就会直接访问主存，从主存中读取到正确的值。在写直达Cache中，有效的Cache块都与主存保持一致。

1. **对同时多线程（Simultaneous Multithreading）、粗粒度多线程和细粒度多线程比较，他们各有什么优缺点。（2006，2010，2013）**
2. 粗粒度多线程：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：大大减少了线程间切换的次数，并且不减慢每个独立线程的执行。

缺点：不能有效减少吞吐率。

1. 细粒度多线程：每条指令之间都能进行程序切换，从而消除了完全空闲的流出槽，导致多个线程交替执行。

优点：能够隐藏由任何或长或短的阻塞带来的吞吐率的损失。

缺点：减慢了每个独立线程的执行。这是因为即使没有任何阻塞，线程也会被其他线程的指令插入执行而导致延迟。

1. 同时多线程：一个时钟周期内调度多条指令使用流出槽。

优点：同时实现指令级和线程级并行。

缺点：实际中一些因素，如活跃线程个数、和缓冲大小限制、可并行指令，从多个线程中取出指令的能力等，仍将限制流出槽的利用率。

1. **在基于总线的集中共享多处理机系统中Cache相关性（Coherency）监听协议原来有3个状态，无效、共享、专有。如果在增加一个干净专有（只读）状态，给出相比原来3个状态时，现在的相关性监听协议工作原理是什么？增加了那些新的状态变迁？（2006）**

**什么是多处理机的相关性（coherency）和一致性（consistency）？在一个基于总线的集中共享多处理机系统中，Cache相关性监听协议有4个状态：无效、共享、专有和干净专有（只读）。给出相关性的监听协议的工作原理。（2012）**

监听协议：每个Cache块除了包含物理存储期块的数据拷贝之外，还保存各个块的共享状态信息，Cache通常连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。

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”。当其他处理器试图读取“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”。



1. **什么是多处理机的相关性（coherency）和一致性（consistency）？给出解决相关性的监听协议的工作原理。（2003，2010，2014）**

**什么是多处理机的相关性（Coherency）和一致性（consistency）？给出解决相关性的目录协议的工作原理及状态变迁。（2007，2011，2015）**

**1）多处理机的相关性和一致性**

1. 相关性：是指一个数据项在任何时候都能读到读数据项最近被写的值。（确定了读取操作可能返回什么值）
2. 一致性：是指一个处理器何时能够读到另一个处理器最近更新的内容。（确定了一个写入值什么时候被读取操作返回）

[关系：互补的，coherency 确定了向同一存储器位置的读写行为，consistency则确定了有关访问其他存储器位置的读写行为。]

**2）监听协议的工作原理**

监听协议：每个Cache中除了包含物理存储器块中的数据拷贝之外，还保存着各个块的共享状态信息。Cache通常连接在共享数据块的总线上，各个Cache控制器通过监听总线来判断它们是够有总线上请求的数据块。



**3）目录协议的工作原理**

目录协议：物理存储器中，共享数据块的状态均保存在“目录”中。对于每个可以调入Cache的存储器数据块，目录中对应一个目录项。目录协议根据该目录块的信息和当前要进行的访问操作，依次对相应的Cache发送控制信息，并完成对目录项的修改。此外，还要向请求处理器发出响应信息。



[注：教材独立缓存的状态转换图的从“共享”到“已修改”的“CPU写入命中”路径有误，该状态转换发送失效消息，这将使其他所有共享该数据块的CPU失效，然而在目录中没有失效的状态处理该数据块的共享者也没有被更新，将使后续状态转移出错。该路径应该发送写入缺失消息，由目录在状态转移的过程发送失效消息。

**4）目录协议工作原理详细版本**

目录协议的基本点：在每个节点增加了目录存储器用于存放目录，存储器的每一块在目录对应有一项，每一个目录项主要有“状态”和“位向量”两种成分。“状态”描述该目录所对应存储块的当前情况；“位向量”共有N位，其每一位对应于一个处理器的局部cache，用于指出该cache中有无存储块的拷贝。当处理器对某一块进行写操作时，只要根据位向量通知所有具有相应拷贝的处理器进行作废操作。

当对cache进行写时，其必须是专有状态，任何一个共享状态的块在存储器中有其最新拷贝。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。存储块可能在任何节点中均无拷贝，也可能在多个节点中同时存在拷贝且可共享读，或者仅在一个节点中存在唯一拷贝且可专有写。除了每个块的状态外，目录项还可用位向量来记录拥有此块的拷贝的处理器，表示出共享集合。对目录表的请求处理需要更新共享集合，作废操作时也要读取这个集合。

目录必须跟踪每个cache块的状态，这些状态包括共享、未缓存和专有。在各个状态下接受到请求和相应需要进行的操作：

 1）当一个块处于未缓冲状态时，对此块发出的请求及处理操作为：

读失效——将存储器数据送往请求方处理器，且本处理器成为此块的唯一共享节点，本块的状态转为共享。

写失效——将存储器数据送往请求方处理器，此块成为专有，表示仅存在此块的唯一有效拷贝，共享集合仅有该处理器，这标识出了拥有者。

  2）当一个块是共享状态时，存储器中的数据是当前最新值，因而可产生两个以上相同请求（第三种请求是写命中，这时对共享集合中其他处理器发送写作废消息，且将共享集合设置为仅含有此处理器，本块的状态变为专有。）：

     读失效——将存储器数据送往请求方，并将其加入共享集合。

     写失效——将数据送往请求方处理器，对共享集合中所有的处理器发送写作废消息，且共享集合置为仅含有此处理器，本块的状态变为专有。

  3）当某块处于专有状态时，本块的最新值保存在共享集合指出的拥有者的处理器中，从而有三种可能的目录请求：

     读失效——将“取数据”的消息发往拥有者处理器，使该块的状态变为共享，并将数据送回目录节点写入存储器，进而把数据发送请求方处理器，将请求方处理器加入共享集合。此时共享集合中仍保留原拥有者处理器。

     写失效——本块将有一个新的拥有者。给旧的拥有者处理器发送消息，要求将其数据送回目录节点，从而再送到请求方处理器，使之成为新的拥有者，并设置新拥有者的标志。此块的状态仍旧是专有。

     数据写回——拥有者处理器的cache要替换此块时必须将其写回，从而使存储器中有最新拷贝（宿主节点实际上成为拥有者），此块成为非共享，共享集合为空。

1. **给出同时多线程（Simultaneous Multithreading）的并行工作原理及其在体系结构实现上的基础。（2008）**

1）原理：同时多线程技术是一种在多流出、动态调度处理器上开发线程级并行和指令集并行的改进的多线程技术。同时多线程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，来自各个独立线程多条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，线程级并行和指令集并行被同时开发。

2）结构基础

使用动态调度技术的处理器已经具有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组，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，而不会导致各线程间源操作数和目的操作数的混乱。

通过乱序执行机制，可以很好地利用硬件功能，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，我们只需通过在支持单线程的处理器上，为每个线程的实现设置重命名表，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。

1. **什么是多处理机的相关性和一致性？2008一台超结点的分布共享多处理机，其Cache相关性（coherency）协议是结点内部运行监听协议，结点之间运行目录协议。试述系统解决相关性的工作原理。（2008，2009，2013）**

超结点分布共享多个处理机系统解决相关性工作原理。

监听：每个Cache块除了包含物理存储器的数据拷贝之外，还保存着各个块的共享状态信息。Cache通常连在数据块的总线上，各个Cache控制器通过监听总线来判断他们是否还有总线上请求的数据。

目录：物理存储器中，共享数据块的状态均保存在目录中，对于每个可以调入Cache的存储器数据块，目录中对应一个目录项。目录协议根据该目录块的信息和当前要进行的访问操作，依次对相应的Cache发送控制信息，并完成对目录项的修改。此外，还要向请求处理器发出响应信息。

这台超级结点分布共享多处理机系统中，目录项所保存的共享数据块状态及相关信息，要与每个超结点内各个块所包含的共享状态信息一致。

监听

目录

（另一个版本）

超结点的分布共享多处理机在节点内部运行监听协议，节点之间运行目录协议。

     即在超结点内部每个处理机的cache除了包含物理存储器中的块的数据副本之外，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，各个cache控制器通过监听总线来判断它们是否有总线请求的数据块。

     每个超结点中使用一个目录存储器用于存放目录，存储器的每一块在目录对应有一项，每一个目录项主要有“状态”和“位向量”两种成分。“状态”描述该目录所对应存储块的当前情况；“位向量”共有N位，其每一位对应于一个超结点，用于指出该超结点中有无存储块的拷贝。

     当处理器进行写时，必须先获得总线的控制权，然后将要作废的数据块地址放在总线上。超结点内其他处理机一直监听总线，检测该地址是否存在于它们的cache中。若存在，则作废相应的数据块。同时，向该块所在的目录发消息，目录根据向量位判断该块是否存在于其它的超结点中，若存在，则根据位向量通知所有具有相应拷贝的超结点进行作废操作。超结点在接收到目录发来的写作废消息后，将该要作废的数据块地址放在总线上。当处理器写未命中时，除了将其它处理器对应的cache块作废外，还需要取回该数据块。

     监听过程的实现可利用cache中块的标志位。每个块的有效位使作废机制的实现较为容易。由写作废或其他时间引起的失效处理很简单，只需要将该位设置成无效即可。而对于写，我们希望知道时候别的cache中也有此数据的共享拷贝。因为如果没有别的cache拷贝，则无需将写地址放在写回cache的总线上，这样可降低所用的时间和所需的带宽。为了区别某个数据块是否为共享，需要给每个cache块加上一个特殊的状态位来说明它是否为共享。当对共享块进行写时，本cache将写作废的请求放在总线上，cache块状态由共享变为非共享或专有，同时目录向其它超结点发送写作废消息，目录中该块的状态变为专有。如果另一个处理器要求访问此块，则该块在cache标志位状态会再转化为共享，若是其它超结点中的处理器访问，则目录中的状态变为共享。

       目录必须跟踪每个数据块的状态在超结点中的状况，这些状态包括共享、未缓存和专有。在各个状态下接受到请求和相应需要进行的操作：

1）当一个块处于未缓冲状态时，对此块发出的请求及处理操作为：

读失效——将存储器数据送往请求方超结点，且本超结点成为此块的唯一共享节点，本块的状态转为共享。

写失效——将存储器数据送往请求方超结点，此块成为专有，表示仅存在一个超结点对此块的拥有唯一有效拷贝，共享集合仅有该超结点，这标识出了拥有者。

2）当一个块是共享状态时，存储器中的数据是当前最新值，因而可产生三种请求：

读失效——将存储器数据送往请求方，并将其加入共享集合。

写失效——将数据送往请求方超结点，对共享集合中所有的超结点发送写作废消息，且共享集合置为仅含有此超结点，本块的状态变为专有。

写命中——对共享集合中其他处超结点发送写作废消息，且将共享集合设置为仅含有此超结点，本块的状态变为专有。

3）当某块处于专有状态时，本块的最新值保存在共享集合指出的拥有者的超结点中，从而有三种可能的目录请求：

读失效——将“取数据”的消息发往拥有者超结点，使该块的状态变为共享，并将数据送回目录节点写入存储器，进而把数据发送请求方超结点，将请求方超结点加入共享集合。此时共享集合中仍保留原拥有者超结点。

写失效——本块将有一个新的拥有者。给旧的拥有者超结点发送消息，要求将其数据送回目录节点，从而再送到请求方超结点，使之成为新的拥有者，并设置新拥有者的标志，同时作废旧的拥有者节点中的该数据块。此块的状态仍旧是专有。

数据写回——拥有者超结点中的所有拥有此块的cache全部替换此块时必须将其写回，从而使存储器中有最新拷贝（宿主超结点实际上成为拥有者），此块成为非共享，共享集合为空。

1. **Tell me the advantages and disadvantages of hardware-based speculation（前瞻）。（2010，2015）**

前瞻技术实现的关键是允许指令的乱序执行，但必须顺序确认，它不仅能够得到更高的并行性，还能在处理器未判断指令是否可以执行前就提前执行，以克服控制相关。基于硬件的前瞻和动态调度相结合，可以做到同一种体系结构，但实现不同机器能够使用相同的编译器。

主要缺点是支持前瞻的硬件太复杂，需要大量资源。

1. **大规模机器的同步有哪些软件和硬件支持方法？（2011）**

软件：带指数延迟的旋转锁、软件排队锁、组合树；

硬件：排队锁、硬件原语。

1. **临界区，临界资源以及程序进入临界区的调度原则？（2011）**

临界资源是一次仅允许一个进程使用的共享资源。无论是硬件临界资源，还是软件临界资

源，多个进程必须互斥地对其进行访问。每个进程中访问临界资源的那段代码称为临界区(critical section)。每次只允许一个进程进入临界区，进入后不允许其他进程进入。

程序进入临界区的调度原则：

1. 若有若干进程请求进入空闲临界区，一次只允许一个进程进入。
2. 任何时候，处于临界区的进程不可多于一个。如果已有进程进入自己的临界区，则其他所有试图进入临界区的进程必须等待。
3. 进入临界区的进程要在有限的时间内退出，以便其他进程能够及时进入临界区。
4. 如果进程不能进入自己的临界区，则应该让出CPU，避免进程出现“忙等”现象。

Linux内核中最常见的锁是自旋锁(spin lock)。自旋锁最多只能被一个可执行线程持有。如

果一个执行线程试图获取一个被已经持有（即所谓的争用）的自旋锁，那么该线程就会一直进行“忙循环——旋转——等待锁重新可用”。要是锁未被争用，请求锁的执行线程便能够立刻得到它，继续执行。在任意的时间，自旋锁都可以防止多于一个的执行线程同时进入临界区。同一个锁可在多个位置。

一个被争用的自旋锁使得请求它的线程在等待锁重新可用时自旋（特别浪费处理器时间），这种行为是自旋锁的要点。所以自旋锁不应该被长时间持有。事实上，这点正是使用自旋锁的初衷：在短时间内进行轻量级加锁。