高体**必考**知识点

**概念题**

*还有一部分概念浪费时间没整理。后果自负。*

**程序局部性**：程序总是趋向于使用最近使用过的指令和数据。包括时间局部性：最近访问过的内容很可能被再次访问；包括空间局部性：地址邻近的内容很可能在一定的时间内被连续使用。

**多线程**：在软件或硬件上实现多个线程并行执行的技术，利用多线程来隐藏流水线以及访存延时。通过开发线程级并行来提高单个处理器的吞吐量。

**Basic block**：一段顺序执行代码，除入口外没有其它转入分支，除出口外无其它转出分支。

**Vector chain**：具有先写后读（RAW）相关的两条指令，在不出现功能单元冲突的情况下，可以把功能部件连接起来执行流水处理，从而减少访存次数，以达到加快执行的目的。

在这种机器中，标量、向量寄存器产生的中间值可以被立即使用，从而减少了访存。

**Hard real-time**：是一个刚性的、不可改变的时间限制，它不允许任何超过时限的错误。超时错误会带来损害甚至导致系统失败、或者导致系统不能实现它的预期目标。

**Soft real-time**：是一个灵活的时间限制，它可以容忍偶然的超时错误。失败造成的后果并不严重，只能提供统计意义上的实时。

当超出某一最大时间，会限制一项特定任务的平均时间和实例数。

**Clusters**：Clusters是通过局部网络将桌面电脑或服务器连接起来从而作为一个大型计算机使用。每个节点有自己的操作系统并通过网络协议通信。最大的集群被称为**WSCs**（Warehouse scale computers）,从而成千上完台服务器表现为一台服务器。

**Availability**：可用性，指服务完成与服务中断两种状态时对服务完成的质量。在一个给定的时间间隔内，对于一个功能单元来说，总的可用时间所占的比例。

**Pipeline interlock**：一种硬件结构，流水线互锁检测冒险，在冒险清除前使流水线停顿。对于RAW冒险，可以利用提前(Forwarding)消除，对于提前不能消除的，可以通过interlock的方式消除。

**ILP**：当指令之间不存在相关时，它们在流水线中时可以重叠起来并行执行的，这种指令序列中潜在并行性称为指令级并行。

**等分带宽**：将网络平均分成两半，沿切口的通道边数的最小值。

**虚拟自适应**：将物理通道分成几个虚拟通道，根据虚拟通道的忙闲情况，自适应的选择后续通道。

**Structure Hazards**：硬件资源满足不了流水线中指令的重叠执行的要求，产生资源冲突。

**全局代码调度**：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。其目的是在保证代码段数据相关和控制相关不变的情况下，把一段内部包含控制相关的代码段压缩到尽可能的短。

**软流水**：软件流水技术是一种重组循环体的技术，在软件流水循环（Software pipeline loop）的每一次迭代（即新的重组后的循环）体是由原循环的不同迭代中选出的指令组成的，可以达到消除相关性的目的。软件流水技术是通过对循环重新进行建构，使得每次迭代执行的指令是属于原循环不同迭代过程的，是用来安排循环指令，是这个循环的多次迭代[并行执行](http://baike.baidu.com/view/160033.htm" \t "/home/smher/Documents\\x/_blank)的一种技术。

是一种循环展开技术，又称为符号化循环展开。经过软流水调度后的代码的每一层循环中指令是由原循环中迭代的不同层次中的指令组成。

**网络直径**：网络中任意两个节点间最短路径的最大值。

**虫孔路由**：把消息包分成小片，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。

**超标量**：相对于标量技术，超标量每个时钟周期可以流出多条指令，流出的指令数不定。

**NUMA**：非均匀存储器访问机制，内存访问时间取决于处理器的内存位置。通过为各个处理器提供分离的存储器，增加了访问的带宽。DSM机器，多处理器采用物理上分布的存储器，但是在逻辑上采用一致的空间编址，访问时间取决于数据在存储器中的位置。

**所有者**：拥有数据块的唯一副本的处理器称为该数据块的所有者。

**宿主节点**：is the node where the memory location and directory entry of an address reside.

存放数据的存储单元以及目录项所在的节点。

**节点度**：与节点相连的边的数量。

**Superblock**：只能拥有一个入口，可以拥有多个出口的块。

**Trace Compaction**：把一长串路径上的指令压缩成少数几条长指令并且尽可能早的调度。

**Instruction Windows**: 在乱序前瞻处理器中可以乱序执行的一组指令集。传统上，由ROB中所有的指令组成，有序的进出IW，在指令提交之前，窗口内的指令都是speculation的。

硬件支撑的可以并行运行的指令集。

**栅栏同步**：强制所有到达栅栏的进程进行等待，直到所有的进程全部到达栅栏，然后释放全部进程，从而形成同步。

**非绑定**：预取能返回最新数据值，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。

**Register pressure**：当循环展开时，以及其它大规模程序段的调度时都会引起寄存器的短缺，被称为寄存器压力。由循环展开和指令调度共同作用。

**Remote node**：拥有某个数据块的拷贝,而不论其是占有（唯一拷贝）还是共享。

**写缺失**：写的时候需要同时更新物理内存；如果物理内存地址对应的值不在cache中则发生写缺失

**旋转锁**：一个锁，处理器会以循环的形式一直尝试去获取并旋转它，直到成功。

**Branch delay slot**：关键词：延时分支策略，下一时钟周期。在延时分支策略中，在分支命令的下一拍执行的指令。

**Decoupled Architecture**：在这种计算机中，利用buffer来将取指、解码阶段和执行阶段分离。允许各个部件之间独立运行。

A processor with queues to allow slippage of loads and stores with other functional units is called ~.

**One-bit BPB**：利用分支命令低位地址寻址的缓存，包含了同一分支指令上次的跳转信息。

**Architecture Register**：程序员可以使用的寄存器。

**Pipeline hazards**：下一条指令不能在制定时间内执行。

**ROB**：保存指令在执行完成后到提交之前的计算结果，从而实现指令的重排序；为程序提供虚拟寄存器。

**Data flow**：数据流，指令执行过程中数据的传送。

**Loop Level Pipeline**：通过循环体展开若干次，通过重命名式动态调度来获取更多的并行性。

**Trace’s compaction**：把一长串路径上的指令压缩成少数几条长指令，并且尽可能早的调度。

**Dispatch**：Tomasulo算法中，指令的发射阶段，如果需要的操作数还没在寄存器中准备好，则需要追踪将要产生结果的功能单元。完成寄存器重命名的功能，消除WAW、WAR等冒险，被称为dispatch。

**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时的时钟数，即将相关指令转化为实际无关的指令。

**名相关**：两条指令使用相同的名，但它们之间没有数据流，称之为名相关。

**动态调度**：通过硬件重新安排指令的执行次序，来调整相关指令实际执行时的关系，减少处理机空转。

**记分牌**：在指令动态调度中，通过相应电路检测到指令运行所需的资源满足并没有数据相关就允许指令乱序执行同时记录下这些指令的运行状态。

**BPB**：使用一片缓冲区记录最近一次或几次的分支历史，用分支指令地址的低位来索引，缓冲的存储区为1位的分支历史记录位。

**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内保存起来，缓冲区以分支指令的地址作为标识。

**ROB**：在推断执行的过程中，为了实现乱序执行但顺序确认，故加入了指令确认阶段需要额外的一套硬件缓冲，来保存那些执行完但未经确认的指令及其结果，这个硬件缓冲称为ROB，同时还用它来在推断执行的指令之间传送结果。

**The efficiency:** the percentage of available slots that contained an operation.计算指令在所有指令中所占的百分比。

**Throughput**：一定时间内可以执行的最多指令数，越大越好。

The total amount of work done in a given time

**简答题**

**同时多线程**：是一种在多流出、动态调度的处理器上开发的一种改进的多线程技术，它同时实现指令级、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。

在体系结构上实现的基础：动态调度处理器已经具备开发线程级并行所需的许多硬件设置。具体的，动态超标量处理器有大量的**虚拟**寄存器，可用来保存独立进程的寄存器状态。此外，通过乱序执行，可以很好的利用硬件功能，提高并行率；通过重排序缓存，将来自独立线程的指令以独立的方式进行提交。

在以上的硬件基础上，只需要为每个线程设置重命名表，设置各自的程序计数器PC并为多个线程提供指令确认的能力。

**大规模机器同步硬件和软件方法**：

硬件：硬件排队锁，硬件原语。

软件：软件排队所，组合树，延时等待旋转锁。

**多处理机进行时延隐藏的主要技术及其原理**：

MPP系统中采用了包括数据预取、相关性cache、松弛一致性、多现场集中时延隐藏技术。

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和延时；

松弛一致性：在保证程序正确性的前提下，增加指令执行的并行。

多现场：当产生时延时进行现场切换转而执行其它程序。

**多指令流出受哪些方面的限制**：

程序所固有的指令级并行性；

硬件实现上的困难；

超标量和超长指令字处理器固有的技术限制。

**简述Tomasulo算法的核心及其与记分牌算法的异同**：

相同之处：

两者都消除了RAW冒险，Tomasulo算法使用了记分牌算法的动态调度的核心思想，多条执行处于发射状态，等待条件成熟，可以不按顺序执行。

Tomasulo算法的核心：

记录和检测指令相关，指令一旦就绪就立即执行，把发生RAW冲突的可能性减少到最小；利用保留站实现寄存器的换名，消除WAR, WAW冒险。

与积分牌的不同点在于：

利用保留实现寄存器换名，从而消除WAW，WAR冒险。而记分牌检测到这两种冒险后需要停顿。

冲突检测和指令控制机制分开，通过分布在功能单元的保留占来进行是分布式的。

计算结果通过数据总线CDB放入保留站进行缓冲。

**比较同时多线程、细粒度多线程和粗粒度多线程，他们各有什么优缺点**？

细粒度多线程可以在线程的每条指令切换到其它指令；由于在每个时钟周期都切换线程，使得多个线程的指令执行过程交织在一起。优点：可以隐藏因为任何长短停顿而导致的吞吐量损失。缺点：使得单个线程的性能降低，这是由于没有停顿的、已经准备好执行的指令可能会被其它线程的指令插入执行而导致延时。

粗粒度多线程仅在停顿较长时才切换到另一个进程；优点是降低了线程切换的成本，很大程度上避免了降低处理器速度的问题。缺点：克服吞吐量损失的能力非常有限，特别是由于较短停顿导致的吞吐量损失，这是由粗粒度多线程的流水线启动开销导致的。

同时多线程使用多发射和动态调度处理器同时实现了线程级并行和指令级并行，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同时发射，而不用考虑指令集间的相关性；相关性由动态调度来处理。在SMT中，每拍有多个指令槽，可以安排多线程的多个指令同时流出。

他们都是硬件自动进行调度，并行性是同时多线程最好，然后是细粒度多线程。

**相关性和一致性**？

相关性：是指一个数据项的任何读均可得到数据项最近被写的值，解决的是内容的正确性问题；

一致性：一个处理器何时读到另一个处理器最近更新的内容，解决的时序的正确性问题。

追踪数据相关性的途径有基于目录的和监听。

保证相关性的方法有写作废、写广播协议等。

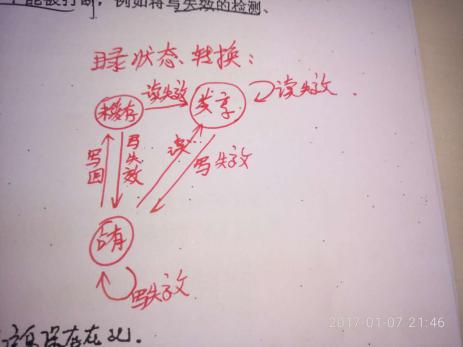
*感觉应该只要记一下目录协议的状态转化图即可，监听协议的太复杂了。*

**目录协议**：

物理存储器的任何一个特定块的共享状态都保存在目录中。

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中都有对应的一项。当对cache写时，其状态必须是专有状态，任何一个共享状态的块在存储器中都有其最新拷贝。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有唯一的拷贝且可专有写。除了每个块的状态外，目录项还可以利用位向量来记录拥有此块的处理器，表示共享集合。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、未缓冲和专有。在各个状态下接收到的请求和相应需要进行的操作（可见下面的状态转换图）：



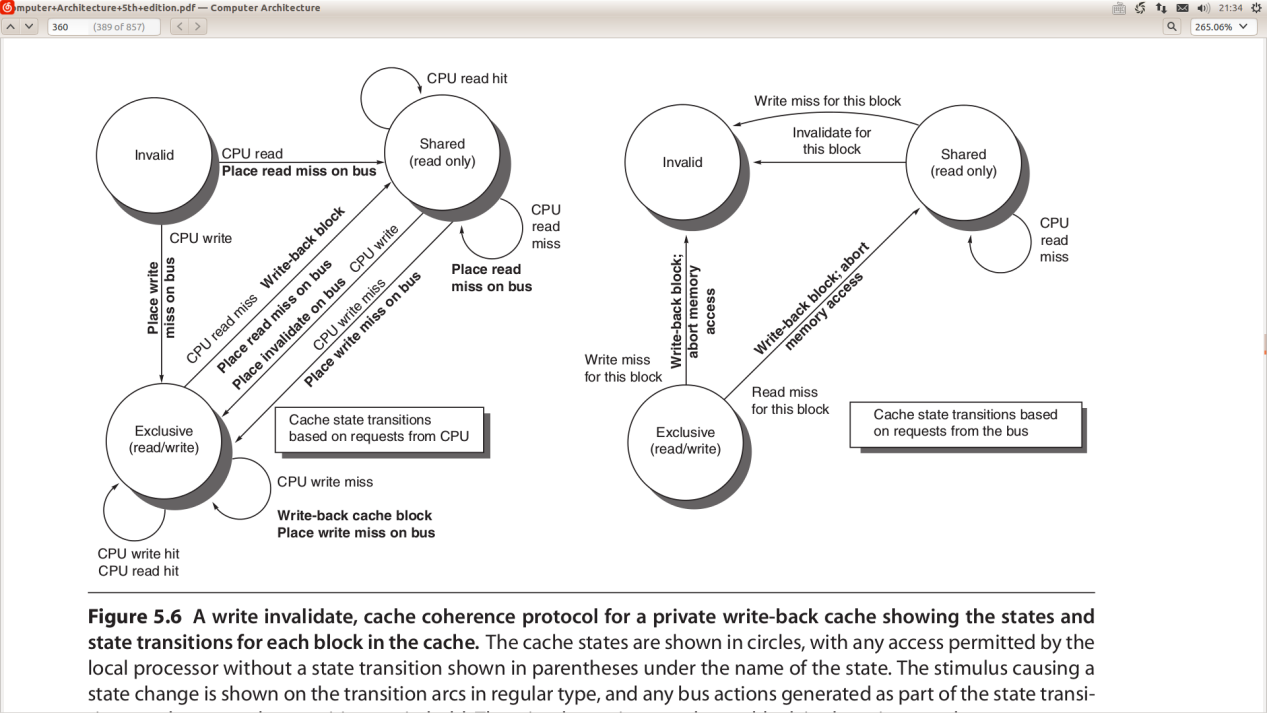
**监听协议**：

每一个拥有数据块副本的cache都需要追踪该数据块的共享状态。

监听协议的关键是对共享数据写的处理。当某个处理器要写数据时，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。其它处理器一直在监听总线，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。获取总线控制权的顺序保证了写的顺序性。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相应的拷贝作废，从而保证了写的严格顺序性。当写cache未命中时，除了将其它处理机上对应的cache数据块作废外，还要从存储器取出该数据块。

监听过程的实现可以利用cache中块的标志位。每个块的有效位使得作废机制实现较为容易。由写作废或其它事件引起的失效处理很简单，只需将其标志位置为无效即可。而对于写，我们希望知道别的cache中也有此数据的共享拷贝。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，这样可以降低所用的时间和所需的带宽。

为了分辨某数据块是否有其它共享，需要给每一个cache块加上一个特殊的状态位来说明它是否共享。当对共享块进行写时，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，则状态再转化为共享状态。



**松弛一致性的四种模型**：

完全存储排序模型：取消W->R顺序。这种模型采用写缓存，并提供读旁路机制，从而允许处理机在其写操作被所有别的处理机看到之前就继续运行读操作；在硬件支持上为写缓存读旁路等。

部分存序模型：取消W->W顺序。允许非冲突写隐含地乱序进行；在硬件上支持为写流水线或其它写并行等。

弱排序模型：取消R->W, R->R顺序。在硬件支持为不封锁读。

释放一致性模型：这种模型区分同步操作中访问一个共享变量的获取操作SA和将对象释放的操作SR。硬件上支持为不封锁读、旁路、无序写等。

**简单比较动态网络中总线、多级网络、交叉开关的特点**

**各自特点**：

总线结构：通过共享总线把各个处理器、共享存储器链接起来，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，但争用严重。

多级网络：每一级用了多个开关，相邻级开关之间都由固定的级间开关。

交叉开关：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与任何一个存储器链接的开关，属于无阻塞网络。

**优缺点比较**：

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。

交叉开关的硬件复杂性以n^2倍上升，所以其造价最昂贵。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。

多级网络则是两种极端的折中。它的优点在于采用模块化结构，因此可扩展性好。但其时延随级数的上升而增加。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。

**写直达cache与写回cache区别**：

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强制写回主存，也不在访问其它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块都与主存数据保持数据一致性。

写直达法：写cache时，cache与主存同时发生写修改。写直达法用来保护cache与主存的一致性，但是当有多个cache都拷贝这个数据时，更新这个cache和主存时会造成其它的cache中的数据不一致。

写回法：写回法cache中的数据会与主存不一致。为了识别cache中的数据是否与主存中的一致，cache中的每一块要增加一个记录信息位，以反映此行是否被CPU修改过。修改cache中某一块时设置这个位为浊。根据这个位的值，cache中每一块都有两个状态：清（clean）和浊（dirty），在将新的值放入浊的块的时候，将原值写回到主存，否则，直接将新值存入这个块。

**集中式共享存储器体系结构中两种维护cache一致性的两种协议**

写作废协议：当一个处理器写某项数据之前保证它对该数据块有唯一的访问权。同时作废其它副本。

写更新协议：当某一个处理器写某项数据时，通过广播使其它cache中所有对应的数据块拷贝进行拷贝。

性能比较：

对同一数据的多个写而中间无读操作的情况，写更新协议需要多次写广播操作，而写作废只需要一次作废操作。

对同一块中多个字进行写，写更新协议对每个字都要广播一次，而写作废协议下仅需要对本块第一次写时进行作废操作。

从一个处理器到另一个处理器读之间的延时通常在写更新模式中较低，而在写作废中需要读一个新的拷贝。

Warehouse-Scale计算机

仓库规模计算机，与云计算相关。

**BTB的工作流程**：

—如果当前指令的地址与缓冲区中的标示匹配，那么此指令必为分支转移成功指令，且下一条指令的PC值在分支目标缓冲的分支PC域中，因此在本指令指令译码阶段开始从预测指令的PC处开始取下一指令；

—如果没有匹配而指令进行转移，则分支目标地址会在ID阶段未知，将其本身的地址及其目的地址加入缓冲区中；

—如果在分支目标缓冲区找到了当前指令地址,而指令当前分支不成功，则将此项目从分支目标缓冲中删去。

—可以看出，如果是分支指令，并且预测正确则不会有任何延时；

—如果预测错误，则会消耗一个时钟周期来取错误的指令，并在一个时钟周期后重新取正确指令；

—如果转移指令不在缓冲区中，则将其当成是不成功的分支指令处理，耗费延时的大小取决于指令是否转移成功。

**动态调度优缺点**：

优点：

可以处理一些在编译阶段无法预见的相关情况；

简化了编译器的设计；

允许在别的流水线机器上编译的指令在不同的流水线上有效的执行。

缺点：

硬件复杂度的显著增加。

**来自书本的内容**：

托马索罗算法的两个主要优点：

1. 分布式的冒险检测

由分布的RS以及CDB完成，通过CDB可以更新多个等待同一结果的指令。

1. 消除了WAW以及WAR

通过寄存器重命名完成。

**必考的设计题**：

1. 计算题：加速比；吞吐量；效率；分支延时等；
2. 循环展开，调度等；
3. 存储器对齐；
4. 检查相关等；
5. 记分牌、Tomasulo、前瞻等算法在指令执行过程中寄存器、RS、ROB的状态转变等。
6. 对于GPU部分，检查循环的同一次迭代中，stored to an array element with index value a\*i + b, and loaded from the same array with index value c\*i+d, 其中i为本次迭代的迭代数。满足以下两个条件时，则存在相关：
7. indices i &j, there are both within the limits of the for loop.
8. a\*j+b = c\*k+d;

As an example, a simple and sufficient test for the absence of a dependence is the greatest common divisor (GCD) test. It is based on the observation that if a loop-carried dependence exists, then GCD (c,a) must divide (d − b).

**Use the GCD test to determine whether dependences exist in the following loop**:

*for (i=0; i<100; i=i+1) {*

*X[2\*i+3] = X[2\*i] \* 5.0;*

*}*

**Given the values a = 2, b = 3, c = 2, and d = 0, then GCD(a,c) = 2, and d − b = −3.**

**Since 2 does not divide −3, no dependence is possible.**

**记得考过的题(英文版教材)：**

1. **P200， Example**

**2. GPU带宽以及吞吐量的题**