Cache一致性问题分析与解决

计科071　　殷成芳
文德春

2010年10月20日

高速缓存（Cache）技术的发明和使用是计算机科学发展史的重大发展，对于提高计算机系统性能起到了重要的作用。“没有高速缓存的586计算机性能比不上有高缓存的486计算机”这种说法可能只是一种理论分析后的简单比较，缺乏证据和说服力。但是 Pentium II（奔腾二代）300CPU与 Celeron（赛扬）处理器的根本差别就是Celeron CPU中没有高速缓存，其性能的差别是显而易见的（当然从价格上，Celeron CPU的价格也比同等 Pentium II CPU的价格低得多）。事实证明，Intel的 Celeron CPU是一个不成功低端策略。在此后的Celeron 300A CPU中，就重新加入了高速缓存，实际测试的结果，Celeron 300A的性能有了大幅度的提高。性能提高的基础和前提是正确性，由于高速缓存本身的特点，存在产生错误的因素，即一致性题，所以只有解决了高速缓存的一致性问题，才能发挥高速缓存的作用，这将是本文关注的重点。为此，我们通过以下三个方面了解Cache及Cache的一致性问题。

一、高速缓存的概念及其他相关概念的介绍（单处理器环境下）；

二、并行系统中的高速缓存一致性问题和解决方法；

三、其他相关、相近概念和内容；

四、小结。

一、高速缓存的概念及其他相关概念的介绍（单处理器环境下）

- 1-1、局部性现象和局部性定义
- 1-2、高速缓存定义
- 1-3、 高速缓存Cache的出现基于的因素
- 1-4、高速缓存的组成结构
- 1-5、高速缓存（Cache）的基本原理
- 1-6、替换算法

1-1、局部性现象和局部性定义

高速缓存的提高系统性能是利用了局部性现象，这里首先对局部性现象和局部性定义加以说明：

对于大量典型程序的运行情况分析结构表明，在一个较短的时间间隔内，地址往往集中在存储器逻辑空间的很小范围内。程序地址的分布本来就是连续的，在加上循环程序段和子程序段要重复执行多次，因此，对程序地址的访问就很自然地具有相对集中的倾向。数据分布的这种集中的倾向不如指令明显，但对数组的存储和访问以及工作单元的选择都可以使存储器地址相对集中。这种对局部范围的存储器地址的频繁访问，而对此范围以外的地址则访问甚少的现象就称为程序访问的局部性。

1-2、高速缓存定义

那么什么是高速缓存、它具有什么特点，如何发挥作用呢？高速缓存的定义如下：

根据局部性原理，可以在主存和 CPU之间设置一个高速的、容量相对较小的存储器，如果当前正在执行的程序和数据存放在这个存储器中，在程序运行时，不必从主存储器取指令和取数据，只要访问这个高速存储器即可，所以提高了程序运行速度，这个存储器称作高速缓冲存储器（Cache：英文原意为“藏东西的地方”）。

高速缓存存储器介于 CPU和主存之间，它的工作速度数倍于主存，全部功能由硬件实现，并且对程序员而言是透明的。

1-3. 高速缓存Cache的出现基于的因素

Cache的出现是基于两种因素：首先，是由于CPU的速度和性能提高很快而主存速度较低且价格高，第二就是程序执行的局部性特点。因此，才将速度比较快而容量有限的SRAM构成Cache，目的在于尽可能发挥CPU的高速度。很显然，要尽可能发挥CPU的高速度就必须用硬件实现其全部功能。

Cache与主存之间可采取多种地址映射方式，直接映射方式是其中的一种。在这种映射方式下，主存中的每一页只能复制到某一固定的Cache页中。由于Cache块(页)的大小为16B，而Cache容量为16KB。因此，此Cache可分为1024页。可以看到，Cache的页内地址只需4位即可表示；而Cache的页号需用10位二进制数来表示；在映射时，是将主存地址直接复制，现主存地址为1234E8F8(十六进制)，则最低4位为Cache的页内地址，即1000，中间10位为Cache的页号，即1010001111。Cache的容量为16KB决定用这14位编码即可表示。题中所需求的Cache的地址为

Cache中的内容随命中率的降低需要经常替换新的内容。替换算法有多种，例如，先入后出(FILO)算法、随机替换(RAND)算法、先入先出(FIFO)算法、近期最少使用(LRU)算法等。这些替换算法各有优缺点，就以命中率而言，近期最少使用(LRU)算法的命中率最高。

cache是一个高速小容量的临时存储器，可以用高速的静态存储器芯片实现，或者集成到CPU芯片内部，存储CPU最经常访问的指令或者操作数据。

1-4、高速缓存的组成结构

高速缓冲存储器是存在于主存与CPU之间的一级存储器， 由静态存储芯片(SRAM)组成，容量比较小但速度比主存高得多， 接近于CPU的速度。

主要由三大部分组成：

Cache存储体：存放由主存调入的指令与数据块。

地址转换部件：建立目录表以实现主存地址到缓存地址的转换。

替换部件：在缓存已满时按一定策略进行数据块替换，并修改地址转换部件。

1-5、高速缓存（Cache）的基本原理

CPU与cache之间的数据交换是以字为单位，而cache与主存之间的数据交换是以块为单位。一个块由若干定长字组成的。当CPU读取主存中一个字时，便发出此字的内存地址到cache和主存。此时cache控制逻辑依据地址判断此字当前是否在 cache中：若是，此字立即传送给CPU；若非，则用主存读周期把此字从主存读出送到CPU，与此同时，把含有这个字的整个数据块从主存读出送到cache中。由始终管理cache使用情况的硬件逻辑电路来实现LRU替换算法 。

下面我们先来看看以上内容中所出现的替换算法

1-6、替换算法

当新的主存字块需要调入高速缓存存储器而它的可用位置又已经被占满时，就产生替换算法问题。常见的替换算法有先进先出（ FIFO）算法、近期最少使用（ LRU）算法、分段LRU算法（ Segmented LRU），工作集 LRU（Worset LRU）基于次数的替换算法（ LFU）和随机替换法（ RAND）。

FIFO算法总是把一组中最先使用的字块替换出去。它不需要随时记录各个字块的使用情况，所以实现容易，开销小。

LRU算法是把一组中近期最少使用的字块替换出去。这种替换算法需随时记录高速缓存存储器中各个字块的使用情况，以便确定近期最少使用的字块。这种算法利用了访问的时间局部性，即如果最近访问了某块数据，该块将会在短时间内很可能被再次访问；如果近期没有访问某个数据块，该块很可能在短期内不会被访问。LRU替换算法的平均命中率比 FIFO要高，并

且当分组容量加大时，能提高 LRU替换算法的命中率。这种算法简单有效，广泛应用于商业文件系统之中。

分段 LRU算法是建立在.. LRU算法基础上的替换算法。这种算法基于以下思想，即被访问超过一次的块很可能被再次访问，因此应该尽量避免替换被访问次数超过一次的块。在分段 .. LRU算法中，缓存分成两个部分，探测段（.. Probe Segment）和保护段（.. Protected Segment）。在这两个段中，数据按照最近被访问的时间排序。当将一个新块存入缓存时，由于该块还没有被访问超过一次，该块要被存放在探测段尾部；如果用户访问的块是已在缓存中的块，由于该块被访问的次数已经超过一次，该块将被移到保护段的尾部。当保护段满时，被从保护段舍弃的块要被作为最近被访问的块放入探测段的尾部。利用这个结构，在选择被替换块时，就能直接选择探测段的头部的块作为被替换块。

分段 LRU算法基于次数的替换算法 LFU是选择缓存中被访问次数最少的块为被替换块的替换算法。这种算法中，要为每个缓存块维护一个计数器，记录该块的被访问次数。在替换时，选择被访问次数最少的块进行替换。这种算法有两个缺点：第一，维护和利用访问次数比访问时间困难的多；第二，当某些短期内被访问多次，却在以后不再被访问的块会长期占据缓存，降低缓存的利用率。为了解决这些问题，LFU算法在实际应用中要进行适当的改进。

以上介绍的替换算法在替换时都只是利用缓存块被访问的状态信息（如访问时间、访、问次数等），没有考虑这些访问可能来自不同的程序，而且这些不同的应用程序可能具有不同的数据访问特征，因此这些算法存在如下几个问题：

1、某些应用程序的访问局部性比较差，如果缓存这些访问应用访问的块，会将其它应用的具有较高缓存价值的块替换掉，降低缓存的有效性。例如，一些大型文件系统的顺序访问可能会将缓存中所有的块都替换出缓存，而代以将来可能不会

再被访问的这些大型文件系统的顺序访问块。这些块占用缓存空间，会阻碍其他重要的数据进入缓存，因此降低了缓存的有效性。

2、对于某种应用有效的替换算法可能对其他应用并不有效。如果缓存系统允许应用程序定义哪种替换算法，就可以使应用程序能够更有效地使用缓存。

3、某些应用程序可以提供将来的块访问信息，但以上讨论的替换算法都没有考虑使用这些信息。如果能够在替换时使用这些信息，可以实现更准确的替换，从而提高缓存的利用率。

4、以上的算法在选择被替换块时没有考虑到公平性。如果某个程序快速地对大量数据块进行访问，会使缓存区中所有块都属于该程序，从而会影响其他程序的运行。

工作集 LRU是一种较好解决以上问题的缓存管理算法。工作集.. LRU为每个执行程序分配一个单独的内存区域，在每个区域内部使用.. LRU算法，每个程序只替换自己的页面。分配给程序的区域大小由系统来动态调整。这种方法具有以下优点：

1、可以将更多的内存分配给具有较高访问局部性的应用程序，以适应这些应用程序的访问特征；

2、对某些区域可以使用有应用指定的替换策略而不使用LRU，这样用户可以将应用指示结合到缓存管理中。

3、通过有系统为每个程序分配相应的系统内存来保证公平性。

但是工作集 LRU也具有一些缺点，如果分给每个应用程序的区域调整得太慢，许多应用程序可能无法利用动态多程序环境中得可用内存：如果调整得太慢，会导致过多得系统调整开销，超过工作集 LRU法所带来的性能改善。

随机替换法（ RAND）不考虑使用情况，在组内随机选择一块来代替。其性能比根据使用情况的替换算法要差一些。

二、并行系统中的高速缓存一致性问题和解决方法

2-1、并行系统的特点；

2-2、Cache一致性问题的发现；

2-3、分析Cache的一致性问题；

2-4、产生高速缓存（ Cache ）不一致的三个原因；

2-5、解决高速缓存一致性的两种协议。

2-1、并行系统的特点

与单机系统相比，并行系统具有自身的显著特点：

（1）具有多个 CPU，同一时刻可以有多个进程同时进行；

（2）各个处理机具有共享内存或私有局部内存，或两者兼备；

（3）各个处理机具有本地高速缓存；

（4）各个处理机之间通过共享总线或交换网络进行通讯，交换数据；

所以除了主存与高速缓存之间可能产生不一致的情况之外，高速缓存与高速缓存之间也可能存在不一致的情况。

2-2、Cache一致性问题的发现

本项目的目标板为：处理器采用ARM芯片S3C44B0X，存储器采用2片Flash和1片SDRAM,在调试的时候输入采用键盘，输出采用显示器，用RS232串口实现通信。

在项目的开发过程中，经软件仿真调试成功的程序，烧入目标板后，程序却发生异常中止。通过读存储器的内容发现，程序不能正常运行在目标板上，是因为存储器中写入的数据与程序编译生成的数据不一致，总是出现一些错误字节。

经过一段时间的调试发现，只要在程序中禁止Cache的使用，存储器中写入的数据将不再发生错误，程序可以正常运行，但速度明显减慢。经过分析，认为问题是由于**Cache数据与主存数据的不一致性造成的。**

Cache数据与主存数据不一致是指：在采用Cache的系统中，同样一个数据可能既存在于Cache中，也存在于主存中，两者数据相同则具有一致性，数据不相同就叫做不一致性。如果不能保证数据的一致性，那么，后续程序的运行就要出现问题。

2-3、分析Cache的一致性问题

要解释Cache的一致性问题，首先要了解Cache的工作模式。Cache的工作模式有两种：写直达模式（write?through）和写回模式（writeback）。写直达模式是，每当CPU把数据写到Cache中时，Cache控制器会立即把数据写入主存对应位置。所以，主存随时跟踪Cache的最新版本，从而也就不会有主存将新数据丢失这样的问题。此方法的优点是简单，缺点是每次Cache内容有更新，就要对主存进行写入操作，这样会造成总线活动频繁。S3C44B0X中的Cache就是采用的写直达模式（write?through）。在写直达模式下，数据输出时，系统会把数据同时写入高速缓冲存储器Cache和主存中，这样就保证了输出时高速缓冲存储器的一致性。但该模式下，却无法保证输入时的高速缓冲存储器的一致性。

2-4、产生高速缓存（ Cache ）不一致的三个原因