

多核系统中共享 Cache 的冒泡替换算法

林晓敏¹, 桂 婷², 乔福明¹, 胡同森¹

(1 浙江工业大学 计算机科学技术学院, 浙江 杭州 310014; 2 浙江工业大学之江学院, 浙江 杭州 310024)

摘 要: 在多核环境下, 对共享 L2 Cache 的优化显得尤为重要, 因为当被访问的数据块不在 L2 Cache 中时(发生 L2 缺失), CPU 需要花费几百个周期访问主存的代价是相当大的。在设计 Cache 时, 替换算法是考虑的一个重要因素, 替换算法的好坏直接影响 Cache 的性能和计算机的整体性能。虽然 LRU 替换算法已经被广泛应用在片上 Cache 中, 但是也存在一些不足: 当 Cache 容量小于程序工作集时, 容易产生冲突缺失; 且 LRU 替换算法不考虑数据块被访问的频率。文中把冒泡替换算法应用到多核共享 Cache 中, 同时考虑数据块被访问的频率和最近访问的信息。通过分析实验数据, 与 LRU 替换算法相比, 采用冒泡替换算法可以使 MPKI(Misses per Kilo instructions)和 L2 Cache 命中率均有所改善。

关键词: 替换算法; Cache; LRU; 冒泡替换; 多核

中图分类号: TP302

文献标识码: A

文章编号: 1000-7180(2011)04-0118-04

Algorithm of Bubble Replacement in Multi-Core Shared Caches

LIN Xiao-min¹, GUI Ting², QIAO Fu-ming¹, HU Tong-sen¹

(1 College of Computer Science, Zhejiang University of Technology, Hangzhou 310014, China;

2 Zhijing College of Zhejiang University of Technology, Hangzhou 310024, China)

Abstract: The shared L2 Cache optimization is extremely important in multi-core architecture, because the cost that L2 miss led to takes hundreds of cycles to access main memory is quite large. The replacement algorithm is an important factor considered for designing Cache and directly affects the performance of Cache and computers overall performance. Although the LRU replacement algorithm has been widely used in on-chip Cache, but there are also several deficiencies; When Cache size is smaller than program working set, it is easy to lead to conflict miss; LRU Replacement Algorithm not consider the access frequency. Bubble replacement algorithm in this article is applied to multi-core shared Cache; meanwhile take the recent access information and the frequency information of the data block into consideration. By analyzing the experimental data, compared with the LRU replacement algorithm, MPKI and the hit rate of L2 has been improved.

Key words: replacement algorithm; cache; LRU; bubble replacement; multi-core

1 引言

Cache 的功能是提高 CPU 数据输入输出的速率。Cache 中的数据虽然能够被快速的访问, 但是其价格是相当昂贵的, 如果计算机主存用这种 Cache 来代替, 则会大大增加计算机的成本, 通常在 CPU 和主存之间设置小容量的 Cache。在存储系统体系结构中, Cache 是访问速度最快的层次。

Cache 的命中率是重要的性能指标之一。由于

其组织方式不同, 数据命中率也不相同^[1]。因此, 需要考虑 Cache 替换算法。

传统的 Cache 替换算法主要有随机替换算法、LRU 替换算法、MRU 替换算法、LFU 替换算法、FIFO 替换算法 5 种。

(1) 随机替换算法。当被访问的数据块不在 Cache 中时, 随机替换算法随机选择一个数据块替换出 Cache, 即每个数据块被替换的概率是相同的。

(2) LRU(最近最少使用)替换算法。该替换算

法使用最为广泛,其基本思想是当被访问的数据块不在 Cache 中时,将最近最少使用的数据块替换出 Cache. 但是该算法也存在一些缺点,将在第2节中进行描述.

(3) MRU(最近被使用)替换算法. 当发生数据块缺失时该替换算法把最近被访问的数据块替换出 Cache, 往往适用于循环访问模式.

(4) LFU(最少访问频率)替换算法. 当发生数据块缺失时该算法选择最少被访问的数据块替换出 Cache. 对于那些符合不相关访问模型的工作负载来说,该算法具有比较好的效果.

(5) FIFO(先进先出)替换算法. 该算法是将最早进入 Cache 的数据块替换出去.

文中把冒泡替换算法应用到多核架构下来改善 Cache 的性能.

2 背景

随着多核时代的到来,计算机的性能在很大程度上受共享 Cache 的影响. 替换算法是设计 Cache 时考虑的一个重要因素. LRU 替换策略在片上 Cache 中被广泛使用,但是 LRU 替换策略仅仅考虑数据块的最近访问的信息,却不关心数据块被访问的频率,当 Cache 容量小于程序的工作集时,Cache 也会表现出抖动现象^[2](即发生冲突缺失:被替换出 Cache 的数据块被不断的换进换出 Cache),会导致计算机性能下降.

为了克服 LRU 替换策略存在的不足,目前提出了不少的解决方案:伪 LRU 替换算法^[3],仅仅用在低相联度 Cache 中;所光等人针对 Cache 对 IPC 影响,以 IPC 指标为最优目标,提出一种最优共享 Cache 划分的方法^[4];基于对 LRU 的修改的算法,廖鑫提出了一种基于 LRU 算法及访问概率改进的 Cache 方案^[5],来提高 Cache 命中率,增强多核适应性,取得一定的效果. 文中采用的冒泡替换算法是通过同时考虑数据块被访问的频率和访问的最近信息,来提高多核架构下,共享 Cache 的命中率.

3 冒泡替换算法应用于多核

多核架构下处理的工作量大于单核处理的工作量,每个核上的线程对共享 L2 的空间资源需求也不同,这就对共享 L2 提出了更高性能要求. 在多核架构下,多个程序或者线程可能表现出不同的行为特征,在设计多核 Cache 替换算法时,数据块被访问的频率和访问的最近信息应该被考虑在内. 冒泡替

换算法,就同时考虑到了这两个因素.

3.1 冒泡替换基本思想

冒泡替换算法包括插入策略、推进策略和替换策略. 为每个 Cache 组(set)设置一个队列,把这个队列称之为冒泡队列.

插入策略是指一个将要被调入 Cache 的数据块放在冒泡队列的底部,因为它仅仅被访问一次,且小于所有其它的数据块被访问的次数. 所以在冒泡队列中考虑到了被访问数据块的频率. 该插入策略与 LRU 插入策略(把调入 Cache 的数据块放在 LRU 栈的 MRU 位置)有着明显的不同,LRU 插入策略并没有考虑数据块被访问的频率.

推进策略是指当一个数据块命中时,命中的数据块就和其上方相邻的数据块交换其位置. 如果命中的数据块已经是队列的最顶端了,则保持其位置不变. 推进策略移动经常被访问的数据块到冒泡队列的顶端,移动不经常被访问的数据块到冒泡队列的底部. 因为最近被访问的数据块,逐步向冒泡队列的顶端推进(逐渐远离冒泡队列的底部),所以同时考虑到了数据块被访问频率和最近访问信息.

替换策略是指被访问的数据块不在 Cache 中时,被替换出 Cache 的数据块有两种选取方法:选取冒泡队列顶端的数据块作为驱出块;选取冒泡队列的底部数据块作为驱出块. 为每个 Cache 组(set)设置一个组命中标签 H,初始化为 0,当 H 值为 1 时,被替换的数据块从冒泡队列的底部选取;当 H 值为 0 时,被替换的数据块从冒泡队列的顶端选取,并且其余的数据块依次向冒泡队列顶端推进一个位置. 图 1 是多核共享 Cache 中其中一个组采用冒泡替换算法的实例,假设该 Cache 采用组相联映射,相联度为 4.

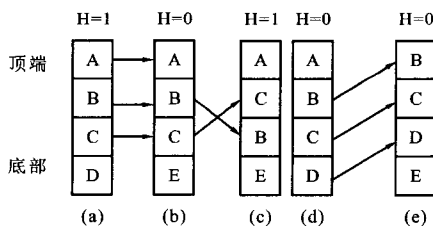


图1 冒泡替换过程实例

在图 1(a)、(b)中,当要访问数据块 E 时,E 不在冒泡队列中,发生缺失,就要从冒泡队列中选择一个数据块替换出去. 因为该 Cache 组的组命中标签 H 为 1,所以从冒泡队列的最底部选取数据块 D 替换出去;当组命中标签 H 为 0 且要访问数据块 E 时,发生缺失,就从冒泡队列的顶端选取数据块 A 替换

出去,其余的数据块 B、C、D 依次向冒泡队列顶端移动一个位置,如图 1 (d)、(e);当冒泡队列中的数据块 C 命中时,数据块 C 就和其上方相邻的数据块 B 交换其位置,如图 1 (b)、(c)。

3.2 冒泡替换算法

多核架构下冒泡替换算法如下:

```

Begin
Set H=0; //组命中标签初始化为 0
IF (Bi miss) //数据块 Bi 不在 Cache 中,发生缺失
    IF H=1
        Remove (top);
        //替换出冒泡队列最顶端的数据块
    ELSE
        Remove (bottom);
        //替换出冒泡队列最低端的数据块
    Rollover ();
    //数据块依次向冒泡队列顶端移动一个位置
Set H=0;
ELSE //数据块 Bi 命中
    IF (i==top) //数据块 Bi 是冒泡队列的最顶端
        Keep(Bi); //保持其位置不变(不移动)
    ELSE
        Exchange(Bi, Bi-1);
        //数据块 Bi 与 Bi-1 交换位置
    Set H=1;
End
  
```

4 仿真实验及其性能分析

实验平台采用的是 Illinois 大学开发的多核模拟器 SESC^[6]. 该模拟器能够用来测试 CMP 系统结构的性能,包括 Cache 命中率、IPC、MPKI 等. 整个实验平台基本配置参数在表 1 中. 片上私有 L1 指令为 32KB 2 路组相联, L1 数据 Cache 为 32kB 4 路组相联. 片上共享 L2 大小为 1M, 8 路组相联. 本实验通过修改模拟器 SESC 中 Cache 设备相关的代码来完成冒泡替换算法的实现.

表 1 Cache 配置

L1 ICache	32KB, 2 路 LRU 替换算法
L1 DCache	32KB, 4 路 LRU 替换算法
L2 Cache	1MB, 8 路 BUBBLE 替换算法

仿真实验中,通过改变 Cache 的大小来测试 L2 的性能: 1M、2M、4M、8M、16M、32M、64M. 测试程序选用的是 SPEC2000, 实验从 SPEC2000 中选取了两个对存储系统要求较高的测试程序: crafty 和 mcf, 可以体现较准确的测试结果.

4.1 对命中率的影响

图 2 显示的是与 LRU 相比采用冒泡替换时命中率提高的百分比. 从图中可以看出, 当 Cache 容量达到 2M 以后, 命中率提高的百分比是一样的, 说明 L2 命中率已经达到最优, 增加 Cache 容量并不能提高 L2 命中率).

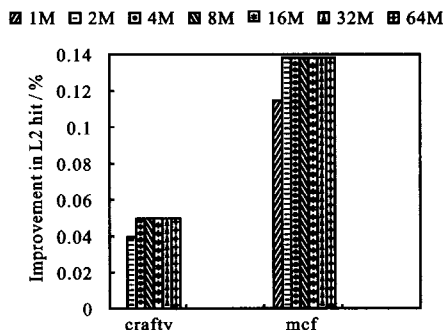


图 2 命中率的提高

4.2 对 MPKI 的影响

每执行一千条指令发生的缺失数(MPKI: Misses per Kilo instructions)是评价 Cache 性能的一个重要因素. 在图 3 中, 对于测试程序 crafty 来说, 随着 Cache 容量的增加, MPKI 随之减少, 但 Cache 容量增加到 8M 以后, 其 MPKI 减少的百分比将保持不变. 但对于测试程序 mcf, Cache 容量的增到 2M 以后, 其 MPKI 减少的幅度不再改变.

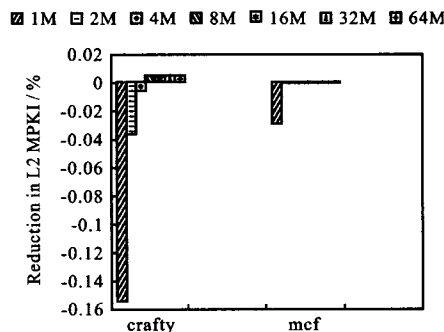


图 3 MPKI 的减少

图 4 显示的是就冒泡替换算法本身而言 Cache 容量对 MPKI 影响. 从实验数据可以看出: 随着 Cache 容量的增加, 其 MPKI 不是一直都是减少的. 对测试程序 crafty 来说, 当 Cache 容量达到 8M 以后, 其 MPKI 减小的幅度将保持不变; 而测试程序 mcf, 达到 2M 以后, 其 MPKI 减少的幅度将保持不变.

4.3 对 IPC 的影响

由实验数据可以看出, 采用冒泡替换算法时, 对于 crafty, 随着 Cache 容量的增加, 其 IPC 也随着增加, 但增加到 4M 以后, IPC 增加幅度将保持不变,

但还是不如 LRU 的性能. 对于测试程序 mcf, Cache 容量增加到 2M 以后其 IPC 增加幅度将保持不变, 其性能与 LRU 相当.

■ 1M □ 2M ▨ 4M ▩ 8M ▤ 16M ▥ 32M ▦ 64M

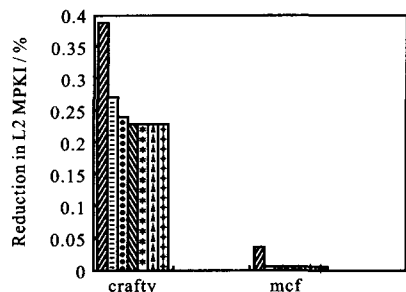


图 4 Cache 对 MPKI 的影响

5 结束语

文中将冒泡替换算法应用到多核共享 L2 Cache 中, 通过分析仿真实验数据, 在小 Cache 容量下算法可以降低 MPKI, 精确度和命中率都比 LRU 有所提高; 但当 Cache 容量达到一定数值后, LRU 算法与文中算法效果相当.

参考文献:

- [1] 戴梅萼. 高性能计算机系统中 Cache 的组织方式及性能分析[J]. 微电子学与计算机, 2000(5): 15-18.

- [2] Wong W, Baer J L. Modified LRU policies for improving second-level cache behavior[J]. HPCA-6, 2000: 49-60.
- [3] Method for implementing a pseudo least recent used (LRU) mechanism in a four-way cache memory within a data processing system[EB/OL](2001)[2010-04-26]. <http://www.patentstorm.us/patents/6240489/description.html>.
- [4] 所光, 杨学军. 双核处理器性能最优的共享 Cache 划分[J]. 微电子学与计算机, 2008, 25(9): 28-30.
- [5] 廖鑫. 一种基于 LRU 算法改进的缓存方案研究与实现[J]. 电子工程师, 2007, 34(7): 46-48.
- [6] Renau J. SESC[EB/OL]. [2010-04-26]. <http://sesc.sourceforge.net>, 2004.

作者简介:

林晓敏 女, (1958-), 实验师. 研究方向为图像处理、嵌入式系统.

桂婷 女, (1974-), 实验师. 研究方向为数字图像处理与模式识别.

乔福明 男, (1984-), 硕士研究生. 研究方向为软硬件节能设计.

胡同森 男, (1954-), 教授. 研究方向为数字图像处理、模式识别.

(上接第 117 页)

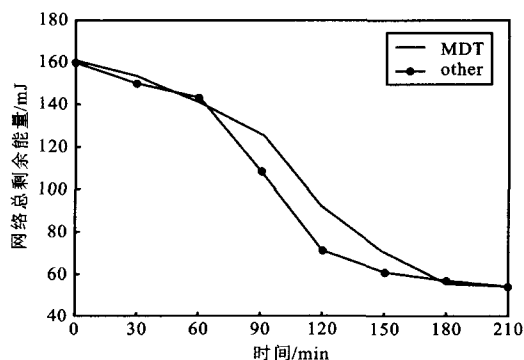


图 2 网络能效比较

参考文献:

- [1] 刘拥民, 蒋新华, 年晓红, 等. 无线传感器网络通信协议[J]. 微电子学与计算机, 2007, 24(10): 71-73.
- [2] Przydatek B, Song D, Perrig A. SIA: secure information aggregation in sensor networks[J]. SenSys, 2003, (3): 255-265.
- [3] Yang Y, Wang X, Zhu S. SDAP: a secure hop-by-hop

data aggregation protocol for sensor networks[J]. MobiHoc, 2006(6): 356-367.

- [4] Hu L, Evans D. Secure aggregation for wireless networks[J]. Workshop on Security and Assurance in Ad hoc Networks, 2003(3): 384-387.
- [5] Ganesan D, Govindan R, Shenker S, et al. Highly-resilient, energy-efficient multipath routing in wireless sensor networks[J]. MobiHoc, 2001(4): 251-254.
- [6] Estrin D, Tutorial. Wireless Sensor networks Part IV: Sensor Network Protocols. MobiCom, 2002[C]// The Eight Annual International Conference on Mobile Computing and Networking Atlanta, Georgia, USA: ACM, 2002: 140-147.

作者简介:

归奕红 女, (1969-), 副教授, 硕士. 研究方向为网络技术、信息安全.

刘宁 男, (1971-), 硕士研究生, 讲师. 研究方向为网络技术、信息安全.