基于IPC与公平性的共享Cache划分

王 震, 徐高潮
(吉林大学 计算机科学与技术学院, 长春 130012)

摘要: 提出一种兼顾高速缓冲存储器(Cache)公平性及系统吞吐率的划分方法, 使用Cache访问监控器记录各应用访问Cache的命中及失效次数, 通过动态划分算法决定每个应用占用的Cache数量, 解决了共享Cache访问冲突导致的Cache污染. 实验表明: 在吞吐率方面, 该方法较传统的LRU替换策略可获得最高37.90%, 平均15.71%的提升, 比公平性最优的划分算法可获得最大47.37%, 平均14.11%的吞吐率提升; 在公平性方面, 较传统的LRU替换策略可获得最大4倍, 平均77%的提升; 比失效率最优的划分算法可获得最大9倍, 平均2.29倍的公平性提升.

关键词: Cache划分; 公平性; Cache访问监控器
中图分类号: TP302 **文献标志码**: A **文章编号**: 1671-5489(2011)04-0740-05

Shared Cache Partitioning Based on IPC and Fairness

WANG Zhen, XU Gao-chao
(*College of Computer Science and Technology*, *Jilin University*, *Changchun* 130012, *China*)

Abstract: This paper proposes a shared Cache partitioning algorithm, based on throughput (IPC) and fairness, which in order to find a balance between fairness and throughput. We used the Cache access monitor to collect the missing rate information, and then decide the amount of Cache resources allocated to each application by the dynamic partitioning algorithm to resolve the problem of Cache pollution. Experiments show that shared Cache partitioning based on IPC and fairness improves throughput by 15.71% on average (up to 37.90%) over least recently used and by 14.11% on average (up to 47.37%) over fairness based Cache partitioning, and improves fairness by 77% on average (up to 4 times) over least recently used and by a factor of 2.29 on average (up to 9 times) over utility based Cache partitioning.

Key words: Cache partitioning; fairness; Cache access monitor

随着微电子技术和硬件技术的发展, 片上多核处理器(CMP)成为当前发展的主流[1-3]. 微处理器可以同时执行多个应用, 这些应用共享L2 Cache, 但传统的最近最少使用(LRU)替换策略并不能区分不同应用, 从而出现Cache污染, 导致系统性能下降及执行时间不确定等问题. Chandra等[4]研究表明, 在双核结构, 二级Cache大小为512 K、8路组相联、每行64字节的配置下, 同时运行两个线程时二级Cache失效率比运行单个线程时平均增加142%, 系统吞吐率(IPC)平均下降23%.

Cache划分技术是解决共享Cache访问冲突的有效方法, 现有的划分方法主要有: 基于公平性的Cache划分(FCP)[5]、以降低Cache访问失效率为目的基于效用度的Cache划分(UCP)[6]和以提高系统吞吐率为目标的Cache划分(ICP)[7]等. 但现有的划分方法并不能很好地平衡系统吞吐率与资源公

收稿日期: 2010-08-25.

作者简介: 王 震(1985—), 男, 汉族, 硕士, 从事分布式计算与微处理器结构的研究, E-mail: wangzhen85@163.com. 通讯作者: 徐高潮(1966—), 男, 汉族, 博士, 教授, 博士生导师, 从事分布式计算与网络软件的研究, E-mail: xugc@jlu.edu.cn.

平性，追求系统吞吐率和失效率最低时可能导致部分线程饥饿，而单纯以公平性为目的的划分将影响系统性能.

本文提出一种兼顾资源公平性与系统吞吐率的 Cache 划分方法(I-F CP)，使用 Cache 访问监控器记录各应用访问 Cache 的命中及失效次数，通过栈距离剖析得到各应用在不同 Cache 容量下的失效率信息，最后使用 I-F CP 计算每个应用应分配的 Cache 容量，得到最优的 Cache 划分. 实验表明，该方法使系统的公平性和吞吐率均有显著提高.

1　基于 IPC 与公平性的 Cache 划分

1.1　系统结构

图 1 为片上多核系统进行 Cache 划分的基本框架(各个核私有的一级数据及指令 Cache 未具体给出). 每个核都增设一个二级 Cache 访存监控器(Cache access monitor，CAM)，用于记录各应用访问二级 Cache 时的命中及失效情况，这些信息经过栈距离剖析后被传送到划分算法中经计算得到最优的 Cache 划分，由于 CAM 并不处于访存的关键路径，因此不会引入额外的时间开销.

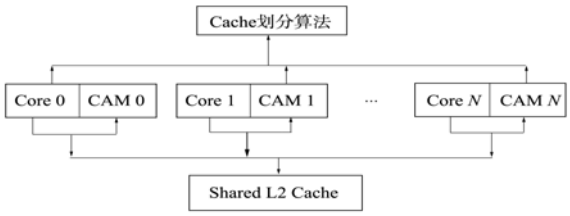

图 1　I-F CP 整体框架

Fig. 1　Framework for I-F CP

1.2　Cache 访存监控器

Cache 访存监控器(CAM)用于记录各应用在所有可能 Cache 容量下的命中及失效信息，其结构基于 Qureshi 等[6]提出的 Utility Monitors，由伪 Cache 块组成，伪 Cache 块包括 Tag 目录等除数据块外与二级 Cache 完全相同的内容，同时采用与二级共享 Cache 完全相同的 LRU 替换策略，可完全模拟各应用独占二级 Cache 时的访存寻址过程. 以 4 路组相连 Cache 为例，如图 2 所示，每个伪 Cache 块根据最近被访问的顺序排列成一个 LRU 栈，同时有 4 个计数器分别记录各路 Cache 的命中情况. 如果 Cache 访问命中其中某一路，则该位置的计数器加 1；如果访问未命中，则专门记录的 Count[miss]加 1，这样就记录了应用访问该 4 路组相连 Cache 的命中及失效次数，从而通过栈距离剖析即可得到该应用在不同 Cache 容量下的失效率信息. 完全复制所有共享 Cache 的伪 Cache 块，需要大量硬件开销，实验证明，通过分组方式只需要复制 32 个伪 Cache 块，即可获得 90% 以上的失效率精度.

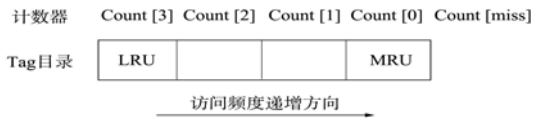

图 2　Cache 访存监控器

Fig. 2　Cache access monitor

1.3　栈距离剖析

栈距离剖析反映应用对 Cache 空间的重用行为[8-9]. 为一个使用 LRU 替换策略的 N 路组相连 Cache 设置 $N+1$ 个计数器，在 Cache 块的 LRU 栈内，各列 Cache 按被访问频次依次排列，被访问次数最多的(MRU)排在第一列. 对于每次 Cache 访问命中，与命中列编号相对应的计数器加 1；若访问失效，则专门记录失效次数的计数器加 1. 栈距离剖析的信息可以通过编译器[8]或仿真模拟获得，本文通过 CAM 模拟独占二级 Cache 得到. 通过以上信息，可易得在更小 Cache 容量下的失效率信息，若分给某应用 $M(M<N)$ 列 Cache，则其失效次数为 $\mathrm{Miss}=\sum_{i=M}^{N}\mathrm{Count}[i]$，其失效率为

$$\mathrm{MissRate}=\frac{\mathrm{Miss}}{\sum_{i=0}^{N}\mathrm{Count}[i]}.$$

1.4　I-F 模型

平衡吞吐率与公平性的度量公式为[10]

$$\mathrm{Fair}_{\mathrm{IPC}}=N/\sum_{i=1}^{N}(\mathrm{SingleIPC}_i/\mathrm{IPC}_i),\tag{1}$$

其中：$\mathrm{SingleIPC}_i$表示应用独占 Cache 空间时的 IPC；IPC_i表示共享 Cache 空间时的 IPC；N 表示应用数目. 在本文算法运行过程中并不能得到准确的 IPC 值，对于典型的共享二级 Cache 体系，IPC 定义为[11]

$$\mathrm{IPC} = \frac{1}{\mathrm{CPI}} = \frac{1}{\mathrm{CPI}[\mathrm{idealCache}] + H_1 + MR_1 \times H_2 + MR_2 \times H_3}, \tag{2}$$

其中：CPI[idealCache]表示目标数据保存在 CPU 寄存器内时的访存开销；H_1, H_2, H_3 分别为访问一级私有 Cache、二级共享 Cache 及内存时的开销；MR_1, MR_2 分别为一级私有 Cache 和二级共享 Cache 的访存失效率. CPI[idealCache]，H_1, H_2, H_3 等值均由机器硬件决定，而 MR_1 可通过为一级私有 Cache 设置计数器获得，MR_2 可通过栈距离剖析获得.

1.5 I-F 划分

假设 M 个相互竞争的应用同时在某 M 核处理器上运行，共享 N 路组相连二级 Cache. 要寻求最优的 Cache 划分，即为获得 $\dfrac{N}{\sum_{i=1}^{N}(\mathrm{SingleIPC}_i/\mathrm{IPC}_i)}$ 的最大值，即寻求 $\sum_{i=1}^{N}(\mathrm{SingleIPC}_i/\mathrm{IPC}_i)$ 的最小值. 本文采用按列逐步划分[12]的方法，即每次循环分配一列 Cache 给其中的某项应用，直到 Cache 资源划分完毕.

假设 $f_i(k) = \mathrm{SingleIPC}_i/\mathrm{IPC}_i(k)$，其中 $\mathrm{IPC}_i(k)$ 为第 i 个应用占用 k 列共享 Cache 时的 IPC，同时定义增益函数 $g_i(k) = f_i(k+1) - f_i(k)$，表示第 i 个应用占据 $k+1$ 列 Cache 时比占用 k 列 Cache 所获得的增益. I-F 划分算法如下：

1) $C = (C_1, C_2, \cdots, C_M) = (1, 1 \cdots, 1)$；//$C_i$ 表示分配给应用 i 的 Cache 列数，算法开始时，保证所有应用均可以获得 Cache 资源；

2) 按照非降顺序对各应用的增益函数 $g_i(C_i)$ 进行排序；

3) for $j = 1, 2, \cdots, N - M$ do；

4) 假设 $g_l(C_l)$ 为队列中最前的一项；

5) $C_l = C_l + 1$；//将本列 Cache 划分给增益最大的应用；

6) 重新按照非降顺序排列各 $g_i(C_i)$；//循环划分，直至所有 Cache 划分完毕；

7) 返回各应用的 Cache 分配情况 C.

2 仿真结果

2.1 仿真环境

仿真实验采用 Virtutech Simics[13]模拟器模拟 Alpha 结构的双核处理器，各处理器拥有私有的一级数据 Cache 及指令 Cache，双核共享二级 Cache，模拟配置参数列于表 1.

表 1 模拟环境配置参数

Table 1 Configuration parameter of simulation

参 数	配置指标
处理器	2.2 GHz, 0.5 ns cycle time, 8-way
一级私有数据、指令 Cache	16 KB, 4-way, 64 B Cache line, LRU
二级共享 Cache	1 MB, 16-way, 64 B Cache line, LRU
主存	2 GB, DDR2, 667 MHz

2.2 测试用例

本文从 SPEC CPU 2000 中挑选部分访存密集型程序，包括 applu, swim, twolf, applu, mcf, parser 等，根据 Qureshi 等[6]的研究结果，本文将这些测试用例分为两类. 第一类包括 applu, mcf, swim，其失效率受所分配 Cache 空间大小的影响并不明显；第二类包括 ammp, twolf, parser，其失效率受所分配 Cache 空间大小的影响较明显. 本文分别设计了 3 种专门的指令流，见表 2.

表 2　3 种多道程序测试用例组合

Table 2　3 Types of benchmarks

分类	编号	用例组合
A 类	1	applu，mcf
	2	applu，swim
	3	swim，mcf
B 类	4	ammp，twolf
	5	ammp，parser
	6	twolf，parser
C 类	7	applu，parser
	8	mcf，parser
	9	swim，twolf
	10	applu，twolf
	11	mcf，ammp
	12	swim，ammp

2.3　仿真结果

本文采用 Cache 动态划分，故每间隔一段时间需要进行 Cache 的重新划分，实验表明，每 600 万时钟周期进行一次再划分为最合理的策略. 图 3 为运用传统的 LRU 替换策略、I-F CP、UCP 及 FCP 方法在给定测试用例情况下的吞吐率测试结果，其中吞吐率度量函数如下：

$$\mathrm{IPC}_{\mathrm{system}} = \sum_{i=1,2,\cdots,N} \mathrm{IPC}_i.$$

由图 3 可见，基于 IPC 与 Cache 公平性的动态划分方法(I-F CP)较传统的 LRU 替换策略可获得最大 37.90%，平均 15.71% 的吞吐率提升；与基于 Cache 公平性的动态划分方法(FCP)相比可获得最大 47.37%，平均 14.11% 的吞吐率提升；与基于失效率的动态划分方法(UCP)相比平均损失 2.15%(最大 10.71%)的吞吐率.

本文使用 Kim 等[5]定义的度量方法考虑公平性，认为理想状态下的最公平情况应为各线程共享 Cache 时执行时间与独占 Cache 时执行时间的比值完全相等，即 $\frac{T_1}{T_{\mathrm{Single1}}} = \frac{T_2}{T_{\mathrm{Single2}}} = \cdots = \frac{T_N}{T_{\mathrm{SingleN}}}$，其中：$T_i$ 表示第 i 个进程在共享 Cache 时的执行时间；$T_{\mathrm{Single}i}$ 表示第 i 个进程在独占 Cache 时的执行时间. 对于这些比值，如果它们的平均差最小，即为最公平. 目前，无法通过具体的数字绝对定义公平性，只能定义相对公平，本文定义相对公平性度量函数如下：

$$\mathrm{Fairness} = \sum_{i=1}^{N} \left| \frac{T_i}{T_{\mathrm{Single}i}} - R_{\mathrm{average}} \right|,$$

其中 R_{average} 表示各比值的平均值. 如图 4 所示，因为完全基于系统公平性的 FCP 算法在计算相对公平值时均为 0，故图 4 中并未给出标识，与传统的 LRU 替换策略相比，I-F CP 可获得最大 4 倍，平均 77% 的公平性提升；与基于 UCP 的划分方法相比，可以获得最大 9 倍，平均 2.29 倍的公平性提升.

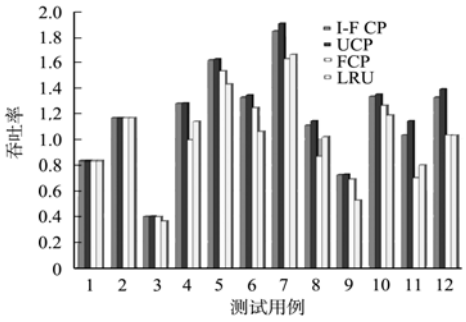

图 3　不同算法的吞吐量

Fig. 3　Throughput of different algorithm

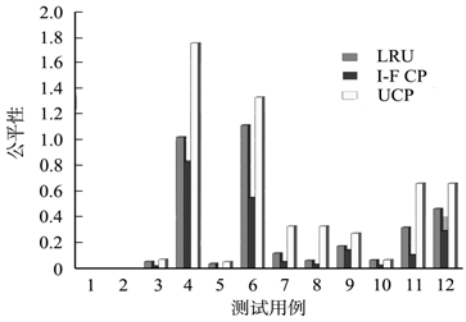

图 4　不同算法的公平性

Fig. 4　Fairness of different algorithm

综上所述,本文提出一种兼顾资源公平性与系统性能的I-F CP算法,解决了现存共享Cache划分方法导致的资源配置不公、系统性能低下等问题. 实验表明,在吞吐率方面,I-F CP较传统的LRU替换策略可获得最高37.90%,平均15.71%的提升;比公平性最优的划分算法可获得最大47.37%,平均14.11%的吞吐率提升;在公平性方面,I-F CP较传统的LRU替换策略可获得最大4倍,平均77%的提升;比失效率最优的划分算法可获得最大9倍,平均2.29倍的公平性提升.

参 考 文 献

[1] Kalla R, Sinharoy B, Tendler J M. IBM POWER5 Chip: A Dual-Core Multithreaded Processor [J]. IEEE Micro, 2004, 24(2): 40-47.

[2] Kongetira P, Aingaran K, Olukotun K. Niagara: A 32-Way Multithreaded Sparc Processor [J]. IEEE Micro, 2005, 25(2): 21-29.

[3] WANG Xiao-yan, LIU Shu-fen, YU Hai. Interface Automata Based Approach to Web Service Composition [J]. Journal of Jilin University: Engineering and Technology Edition, 2009, 39(3): 743-748. (王晓燕, 刘淑芬, 于海. 基于接口自动机的服务组合方法 [J]. 吉林大学学报: 工学版, 2009, 39(3): 743-748.)

[4] Chandra D, GUO Fei, Kim S, et al. Predicting Inter-Thread Cache Contention on a Chip Multi-processor Architecture [C]//Proceedings of the 11th International Symposium on High Performance Computer Architecture. Washington DC: IEEE Computer Society, 2005: 340-351.

[5] Kim S, Chandra D, Solihin Y. Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture [C]// Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques. Washington DC: IEEE Computer Society, 2004: 111-122.

[6] Qureshi M K, Patt Y N. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches [C]//Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. Washington DC: IEEE Computer Society, 2006: 423-432.

[7] SUO Guang, YANG Xue-jun, LIU Guang-hui, et al. IPC-Based Cache Partitioning: An IPC-Oriented Dynamic Shared Cache Partitioning Mechanism [C]//International Conference on Convergence and Hybrid Information Technology. Washington DC: IEEE Computer Society, 2008: 399-406.

[8] Cacaval C, Padua D A. Estimating Cache Misses and Locality Using Stack Distances [C]//Proceedings of the 17th Annual International Conference on Supercomputing. New York: ACM Press, 2003: 150-159.

[9] Cascaval C, DeRose L, Padua D A, et al. Compile-Time Based Performance Prediction [C]//Proceedins of the Twelfth International Workshop on Languages and Compilers for Parallel Computing. London: Springer-Verlag, 1999: 365-379.

[10] Luo K, Gummaraju J, Franklin M. Balancing Throughput and Fairness in SMT Processors [C]//IEEE Intl Symp on Performance Analysis of Systems and Software (ISPASS). Tucson, Arizona: [s. n.], 2001: 164-171.

[11] Matick R E, Heller T J, Ignatowski M. Analytical Analysis of Finite Cache Penalty and Cycles per Instruction of a Multiprocessor Memory Hierarchy Using Miss Rates and Queuing Theory [J]. IBM Journal of Research and Development, 2001, 45(6): 819-842.

[12] LI Zhi, LIU Yuan-ning, WU Ze-xu, et al. Sequence Analysis Based on Dynamic Algorithm [J]. Journal of Jilin University: Information Science Edition, 2010, 28(1): 41-46. (李誌, 刘元宁, 武泽旭, 等. 基于动态算法的序列分析 [J]. 吉林大学学报: 信息科学版, 2010, 28(1): 41-46.)

[13] Magnusson P S, Christensson M, Eskilson J, et al. Simics: A Full System Simulation Platform [J]. IEEE Computer, 2002, 35(2): 50-58.