面向多线程多道程序的加权共享 Cache 划分

所 光 杨学军
(国防科学技术大学计算机学院 长沙 410073)

摘 要 并行应用在共享 Cache 结构的多核处理器执行时,会因为对共享 Cache 的冲突访问而产生性能下降和执行时间不确定的现象.共享 Cache 划分技术可以把共享 Cache 互斥地分配给多个进程使用,是解决该问题的有效方法.由于线程间的数据共享,线程数目不同的应用对共享 Cache 的利用率不同,但传统的以失效率最低为目标的共享 Cache 划分算法(例如 UCP)没有区分应用线程数目的不同.文中设计了一种面向多线程多道程序的加权共享 Cache 划分框架(Weighted Cache Partitioning,WCP),包括面向应用的失效率监控器和加权 Cache 划分算法.失效率监控器以进程为单位动态监控在不同的 Cache 容量下应用的失效率;而加权 Cache 划分算法扩展了传统的失效率最优的 Cache 划分算法,根据应用线程数目的不同在进行 Cache 划分时给应用赋予不同的权值,以使具有更多线程的应用获得更多的共享 Cache,从而提高系统的整体性能.实验结果表明:加权 Cache 划分算法虽然失效率有所增高,但却改进了 IPC 吞吐率、加权加速比和公平性.在由科学和工程计算应用组成的多道程序测试用例中,WCP-1 的 IPC 吞吐率比以失效率最低为目标函数的共享 Cache 划分算法最高高出 10.8%,平均高出 5.5%.

关键词 多核处理器;多线程多道程序;加权共享 Cache 划分;AMRM
中图法分类号 TP302 **DOI 号**: 10.3724/SP.J.1016.2008.01938

A Weighted Dynamic Shared Cache Partitioning Mechanism for Multi-Threaded Multi-Programmed Workloads

SUO Guang YANG Xue-Jun
(School of Computer, National University of Defense Technology, Changsha 410073)

Abstract In a chip-multiprocessor with a shared cache structure, the competing accesses from different applications degrade the system performance, resulting in non-optimal performance and non-predicting executing time. Cache partitioning techniques, a promising solution of the above problems, can exclusively partition the shared cache among multiple competing applications. Processes with different number of threads have different utility on shared cache. However, traditional cache partitioning mechanism, Utility-based Cache Partition (UCP) for example, is to lower the average miss rate of shared cache, regardless of the different thread number of different applications. In this paper, the authors design the framework of Weighted Cache Partitioning, a dynamic shared cache partitioning mechanism to improve the performance of multi-threaded multi-programmed workloads. The framework includes a miss rate monitor, called Application-oriented Miss Rate Monitor (AMRM), which dynamically collects miss rate information of multiple multi-threaded applications on different cache partitions, and weighted cache partitioning algorithm, which extends traditional miss rate oriented cache partition algorithms by adding power coefficient for applications based on their thread number. So the applications with more threads tend to get more shared cache in order to improve the overall system performance. Experiments show that al-

收稿日期:2008-05-31;最终修改稿收到日期:2008-09-05.本课题得到国家自然科学基金(60621003,60603081)、国家"八六三"高技术研究发展计划项目基金(2007AA12Z147,2007AA01Z102)资助.所 光,男,1980 年生,博士研究生,研究方向为多核处理器结构和并行计算.E-mail:suoguang@nudt.edu.cn.杨学军,男,1963 年生,教授,博士生导师,研究领域为并行计算机体系结构、并行计算、高级编译技术、容错计算和操作系统.

though WCP has higher miss rate compared with miss rate oriented cache partition algorithm, it has better IPC throughput, weighted speedup and fairness. Specifically, for multi-threaded multi-programmed scientific computing workloads, WCP-1 improves throughput by up to 10.8% and on average 5.5% over miss rate oriented algorithm.

Keywords multi-core processor; multi-threaded multi-programmed workloads; weighed shared Cache partitioning; AMRM

1 引　言

受限于功耗和复杂度等因素,传统的超标量处理器已经无法有效利用不断增长的晶体管资源.然而集成在片上的晶体管数目仍然会增加,因此多核处理器(CMP)成为今后处理器发展的重要趋势.当前主流的多核处理器如 IBM 公司的 Power 5[1]和 Sun 公司的 Niagara[2]都采用共享最后一级 Cache 的结构.类似于 SMP 结构的多处理机系统,多核处理器中的每个核都执行一个进程或线程,这些进程或线程通常会共享使用片上 Cache 资源.

但是传统的最近最少使用(LRU)的 Cache 替换策略并不区分不同应用的访问,因此一个应用的 Cache 失效可能会替换另一个应用的 Cache 块(又称之为 Cache 污染).所以当多核处理器并行执行多个应用时,应用的执行时间变得不可确定并且依赖于并行执行的其它应用.这导致两方面问题:首先,并行执行的应用对共享 Cache 的冲突访问导致并行执行的应用的性能比独占方式使用 L2 Cache 时下降很多;其次,应用实际拥有的 Cache 会受到并行执行的其他应用的影响,导致应用执行时间的不可预测.研究[3-7]表明,对片上最后一级 Cache 的冲突访问对系统的性能影响很大.因此提高 CMP 系统的性能的关键点之一在于高效管理片上最后一级 Cache 资源.本文研究的问题即为多个多线程应用间共享 Cache 划分问题.

当前共享 Cache 划分领域的工作集中在:以提高系统 Cache 资源划分公平性为目的的 Cache 划分[3];以牺牲系统公平性为代价、以提高系统整体性能为目标的 Cache 划分[4-5];以牺牲系统中某些应用性能为代价,而保证其它某个应用的执行的以服务质量为目标的 Cache 划分[6-7].但是已有 Cache 划分并没有考虑到不同应用线程数目的不同.例如,把单线程应用和 4 线程应用等同对待.正如操作系统需要对线程数不同的应用分配不同的处理机资源,共享 Cache 划分时也应该考虑线程数目的不同.本文研究了线程数目对共享 Cache 划分的影响,改进了传统的 Cache 划分算法,在 Cache 划分时根据应用中线程数目的不同赋予应用一定的权值,从而提出加权 Cache 划分算法,并且从系统最终吞吐率、失效率、加权加速比和公平性的角度评估加权 Cache 算法.

为了能够动态获得应用在不同 Cache 容量下失效率等信息,Qureshi[4]和 Suh[5]分别提出了 Utility Monitor 和 Memory Monitor 以便动态地获取应用在不同 Cache 容量下的失效率.但以上二者都是把线程作为获取的基本单位,只适合单线程类应用.为获取多线程应用在不同 Cache 容量下的失效率,我们设计了面向应用的失效率监控器(Application-oriented Miss Rate Monitor,AMRM).AMRM 具有硬件开销低,预测精度可调,结构化,并且支持多线程应用等优点.

基于 AMRM,我们设计并且评估了加权 Cache 划分算法.根据权重取值的不同,我们设计了三种基于权重的 Cache 划分算法,并且基于全系统模拟器评估了这些算法.实验结果表明:加权 Cache 划分算法虽然失效率有所增高,但却改进了 IPC 吞吐率、加权加速比和公平性.

本文第 2 节提出加权共享 Cache 划分框架、系统结构和算法;第 3 节给出实验平台、度量指标和测试用例;第 4 节给出实验结果和分析结论;最后我们总结全文.

2 加权共享 Cache 划分

2.1 框架介绍

图 1 所示为面向多线程应用的加权共享 Cache 划分框架,我们扩展多核处理器系统结构和操作系统,在处理器中加入面向应用的失效率监控器(AMRM);在操作系统中加入共享 Cache 划分算法

(Cache Partitioning Algorithm);并且给用户提供 Cache 划分支撑工具(Cache Partitioning Utility)以便用户能够控制共享 Cache 的划分策略. AMRM 以应用为单位预测该应用在不同共享 Cache 容量下的失效率. AMRM 的电路结构独立于共享 Cache,便于集成到已有的多核处理器中. 运行于操作系统内核态的 Cache 划分算法根据 AMRM 提供的失效率信息在并行执行的多个应用间动态地调整 Cache 容量的划分. 用户可以使用 Cache 划分支撑工具动态地配置 Cache 划分算法. 我们把 Cache 列(Cache column)[8]作为 Cache 划分的基本单位,需要修改共享 Cache 的替换策略以支持共享 Cache 的划分.

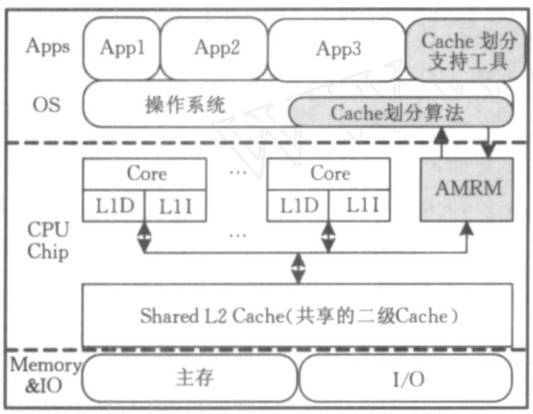

图 1 面向多线程应用的加权共享 Cache 划分框架

2.2 系统结构支持

为使多核处理器硬件支持共享 Cache 的划分,我们采用按照 Cache 列划分的共享 Cache 结构. 我们还修改了 Cache 控制器以便 Cache 的查找和替换操作只在某些 Cache 列内完成. 我们给每个核都添加一个长度为共享 Cache 相联度的比特向量,该比特向量标示共享 Cache 列的使用情况. 所以我们只需要设置比特向量就能够完成 Cache 列的划分. 当 Cache 查找时,所有的 Cache 列都会被查找以避免 Cache 一致性协议的开销和副本问题. 当 Cache 失效时,只能从由比特向量标示拥有的 Cache 列中寻找替换对象. 如果分配给一个核的 Cache 列的数目增加了,那么只有在发生 Cache 失效时,新增的 Cache 才可能被使用. 因为 Cache 失效时,替换对象的选择并不处于访存的关键路径,所以该支持 Cache 划分的体系结构并不引入额外的时间开销.

2.3 AMRM

Qureshi[4]在多核处理器中为每个核都加入一个效用度监控器(Utility Monitor)以记录运行于该核上的应用在所有可能的 Cache 容量下的失效率. 效用度监控器包含伪 Cache 块(伪 Cache 块包含共享 Cache 块除数据外的所有域;和 Cache Set 对应,由伪 Cache 块组成的是伪 Cache Set)的阵列,这样每个核都可以使用私有的 LRU 策略模拟独占使用共享 Cache 时的执行情况. 完全复制所有共享 Cache 的伪 Cache 块会引入很大面积开销. Qureshi 的实验表明只复制 32 个伪 Cache Set 就可以获得 90 %的失效率预测精度. 伪 Cache Set 的数目越多,获得的精度越高. 效用度监控器的缺点是只能监控单线程应用程序的失效率.

基于效用度监控器只需复制少量伪 Cache Set 就可以获得较高精度的思想,我们设计了面向应用的失效率监控器. 其优点为:结构单一,在多核处理器中只需要加入一个 AMRM 模块;失效率精度可调,可监控多线程应用的失效率.

AMRM 所处的位置如图 1 所示,我们不修改处理器访问共享 Cache 的关键路径,因此 AMRM 并不引入额外时间开销. 我们假设每个处理器都有私有的一级 Cache,这些一级 Cache 通过总线互联的方式连接在一个共享的二级 Cache 上. 为区分不同应用的访问,我们为每个处理器加入一个寄存器 AID,它记录了运行于处理器上的线程所在进程的 ID. 当处理器访问一级 Cache 失效时,把 AID 内容和失效地址一起发到总线上. AID 由操作系统在调度线程到处理器执行时设置. 通过监控 L1 Cache 失效时访问 L2 Cache 的地址,并且把地址交给 AMRM 模拟应用独占使用 L2 Cache 时的访问情况,我们就可以得到应用在获得不同 Cache 划分情况下的失效率.

图 2 所示为 4 核处理器的 AMRM 结构,该处理器拥有 4 路组相联共享 L2 Cache. AMRM 由 3 部分组成:地址过滤部件、伪 Cache Set 阵列和计数器阵列. 地址过滤部件包含一个段表,表示应用所使用的伪 Cache Set. 段表由 Start 和 Length 两个域组成,Start 表示该应用所使用的伪 Cache Set 起始地址,Length 表示该应用所使用的伪 Cache Set 长度. 地址过滤部件根据失效地址计算出失效地址在 L2 Cache 所在的伪 Cache Set,然后判断该伪 Cache Set 是否需要模拟执行(判断伪 Cache Set 的索引 Index 是否落在 0 到 Length 之间),如果需要,则模拟执行;否则不做任何动作. 伪 Cache Set(为简化表示,没有标出 Cache Set 的标志位)模拟 Cache 访问的

过程和 L2 Cache 访问的过程相同,也需要进行 Tag 和标志位比较,失效时也需要进行 Cache 替换. 若地址在 AMRM 中不命中,则计数器阵列中由 AID 索引的失效地址计数器 MissCnter 加 1;若命中,则由命中 Tag 对应的 LRU 值和 AID 共同索引的计数器加 1. 图 2 的例子中,应用 2 的失效地址经地址过滤部件产生的索引地址为 $i(64 < i < 128)$,通过在 AMRM 的 LRU 和 Tag 阵列查找,在 Way 1 处命中,相应的 LRU 值为 2,则应用 2 的计数器组的 2 号计数器 Counter2 加 1.

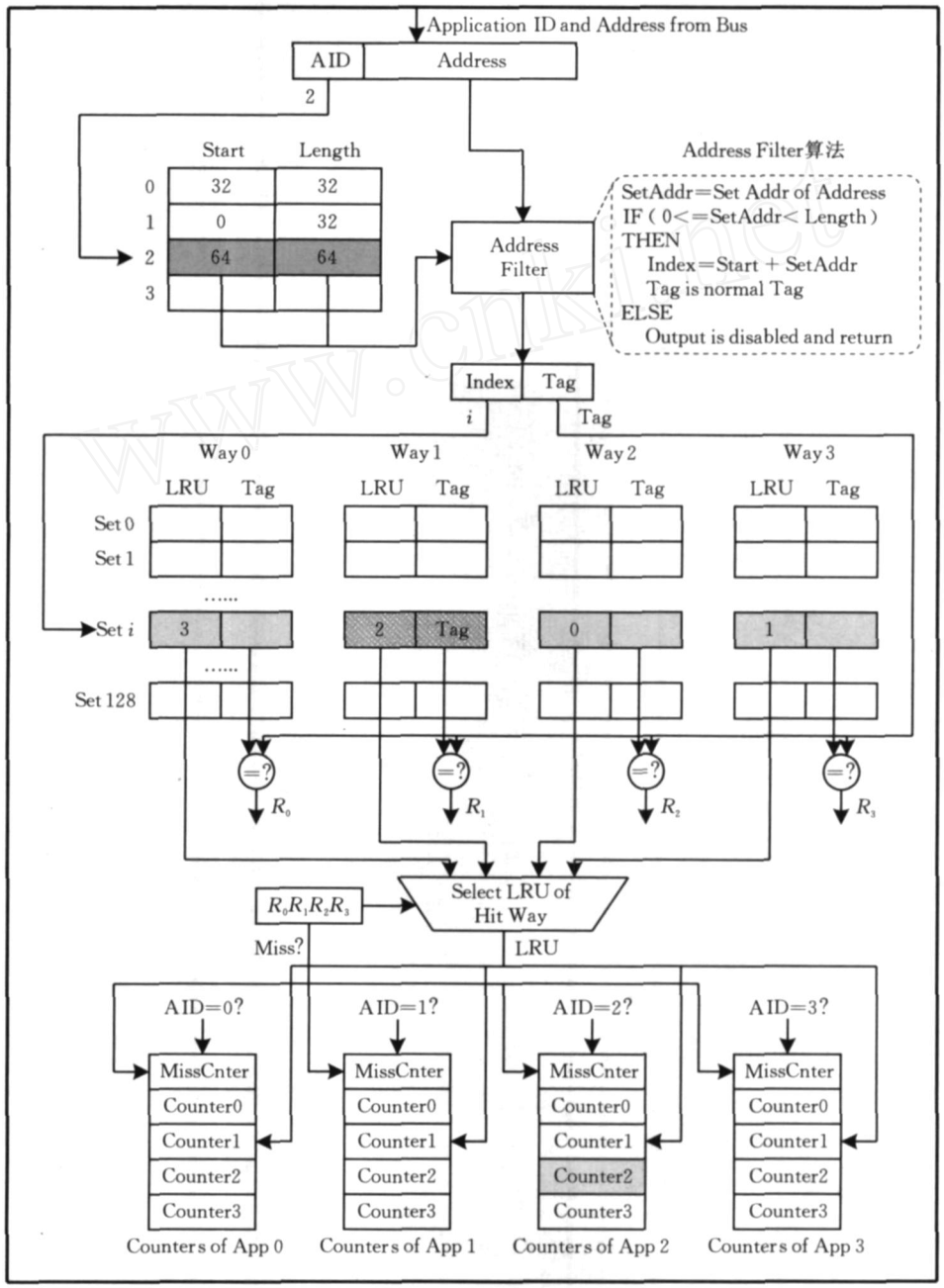

图 2　面向应用的共享 Cache 失效率监控器结构

对于 4 核处理器,AMRM 最多需要监控 4 个应用程序的失效率信息,在 90 %的精度要求下,AMRM 需 128(32 ×4)组伪 Cache Set,若运行多线程应用,则会有伪 Cache Set 空闲,此时可以把空闲的伪 Cache Set 分配给需要的应用,从而获得更高的预测精度. 通过修改地址过滤部件的段表中的 Start 和 Length 域就可以做到分配伪 Cache Set 给应用.

2.4　划分算法

假设 N 核 CMP 处理器中有大小为 C_{L2} 的共享 Cache,有 M 个应用期望到系统中运行,第 i 应用 app_i 需要 p_i 个处理机执行,并且应用集合所需求的线程数目不超过 CMP 处理器拥有核的数目. Cache

划分问题就是给每个应用分配一定的共享 Cache 容量 c_i. 假设 Cache 分配的最小单位为 Δ(Δ 整除 C_{L2}),则容量为 C_{L2} 的 L2 Cache 就可以分为 $CN=\left\lceil\frac{C_{L2}}{\Delta}\right\rceil$ 块. 对于基于列划分的共享 Cache 划分结构,Δ 等于共享 Cache 中一列 Cache 容量. 为简化表示,我们将不区分整数 i 和 $i\cdot\Delta$.

Cache 划分算法通过读取 AMRM 中的计数器,计算进程在不同 Cache 容量下的失效率. 假设 $\lambda_i(k)$ ($k\in[0,CN]$) 为 app_i 在 L2 Cache 容量为 k 时 L2 Cache 的失效率,并且认为 $\lambda_i(k)$ 为关于 k 的非增凹函数,并且 $\lambda_i(0)=+\infty$. 则

$$\lambda_i(k)=\frac{C_{L2}}{Length_i}\cdot\frac{MissCnter_i+\sum_j Counter_{i,j}}{\sum_{1\le j\le p_i} I_{i,j}},$$

其中 $Length_i$ 为 app_i 使用为 Cache Set 的数目; $I_{i,j}$ 为应用 i 的线程 j 执行的指令数; $MissCnter_i$ 和 $Counter_{i,j}$ 为 AMRM 中应用 i 的计数器阵列,$MissCnter_i$ 记录了应用 i 的失效次数,$Counter_{i,j}$ 记录应用 i 的 LRU 值为 j 时的命中次数.

传统的目标为降低平均失效率的 Cache 划分算法等价于求解规划问题:

$$\begin{cases}\text{minimize}\ \sum_i^M \lambda_i(c_i)\\ \sum_i^M c_i=C_{L2}\\ \sum_i^M p_i\le N\\ c_i>0,\ i=1,2,\cdots,M\end{cases}\quad(1)$$

式(1)的解 c_i ($1\le i\le M$) 即为最优的 Cache 划分方案. 其中 $\sum_i^M\lambda_i(c_i)$ 为目标函数,其意义为系统平均失效率最低; $\sum_i^M p_i\le P$ 的意义是任务集合所需求的线程数目不能超过 CMP 处理器拥有核的数目; $\sum_i^M c_i=C_{L2}$ 的意义是为任务集合所分配的 L2 Cache 容量等于 CMP 处理器的 L2 Cache 容量; $c_i>0$ ($i=1,2,\cdots,M$) 的意义为任务集合的每个任务都需分配共享 Cache.

然而式(1)所求解的 Cache 划分并没有考虑系统中不同应用线程数目的不同. 事实上,对于多线程应用,多个线程共享使用分配给应用的 Cache,由于线程间的数据共享,多个线程共同执行时,可以用较低的共享 Cache 失效率获得较高的性能. 所以单纯根据应用的失效率划分共享 Cache 资源是不公平的,具有较多线程的应用应该比较少线程的应用优先获得更多的共享 Cache. 因此我们修改式(1)规划的目标函数,在失效率前加入权重项,权重只和应用的线程数目相关. 变形后的规划问题如式(2)所示:

$$\begin{cases}\text{minimize}\ \sum_i^M w(p_i)\cdot\lambda_i(c_i)\\ \sum_i^M c_i=C_{L2}\\ \sum_i^M p_i\le N\\ c_i>0,\ i=1,2\cdots,M\end{cases}\quad(2)$$

$w(p)$ 为权重函数,p 为应用的线程数. 其中 $w(x)$ 为关于 x 的非降函数,即应用的权值随着线程数目的增加而非降. 借用范数的定义,我们定义了 3 种权重函数启发式:

$$w^0(x)=\|x\|^0=1\quad(3)$$

$$w^{0.5}(x)=\|x\|^{\frac{1}{2}}=\sqrt{x}\quad(4)$$

$$w^1(x)=\|x\|^1=x\quad(5)$$

基于启发式(3)～(5),我们定义了 3 种加权 Cache 划分算法,WCP-0、WCP-0.5 和 WCP-1. WCP-0 的权函数为 $w^0(x)$,对应传统的非加权的 Cache 划分算法;WCP-0.5 的权函数为 $w^{0.5}(x)$,其意义是应用的权等于应用线程数目的平方根;WCP-1 的权函数为 $w^1(x)$,其意义是应用的权等于应用线程数目.

为高效求解式(2)所表示的规划问题,我们定义失效率微增益函数

$$MMR_i(x)=\frac{w^*(p_i)\cdot\lambda(x\cdot\Delta)-w^*(p_i)\cdot\lambda((x+1)\cdot\Delta)}{\Delta},$$

其中,$i=1,2,\cdots,M$, $x=0,1,\cdots,CN-1$, $*=0,0.5,1$. 可证明 $MMR_i(x+1)\le MMR_i(x)$ ($i=1,2,\cdots,M$, $x=0,1,\cdots,CN-1$).

给定向量 $C^k=(c_1^k,c_2^k,\cdots,c_M^k)$,且初始状态为 $C^0=(1,1,\cdots,1)$,对任意的 $K=0,1,2,\cdots$,计算 $i(k)$,使得 $i(k)$ 满足 $MMR_{i(k)}(c_{i(k)}^k)=\max\{MMR_i(c_i^k)\mid 1\le i\le M\}$,那么 C^{k+1} 的计算公式如下:

$$c_i^{k+1}=\begin{cases}c_i^k, & i\ne i(k)\\ c_i^k+1, & i=i(k)\end{cases},\ i=1,2,\cdots,M\quad(6)$$

根据式(6)就可以得到式(2)所代表规划问题的最优解(证明略),Cache 划分算法如图 3.

```
CachePartition(λ)
1. CN = C_L2 / Δ
2. C = (c_1, c_2, …, c_M) = (1, 1, …, 1)
3. sort array MMR = { MMR_i(c_i) | 1 ≤ i ≤ M } by nondecreasing order
4. for k = 1.. CN - M do
5.     assume MMR_i(k)(c_i(k)) is head of array MMR
6.     c_i(k) = c_i(k) + 1
7.     resort array MMR by nondecreasing order
8. return C
```

图 3　基于加权共享 Cache 划分的算法

图 3 所示为共享 Cache 划分算法. CN 为可分配的共享 L2 Cache 块数,初始为 $\frac{C_{L2}}{\Delta}$; C = $(c_1, c_2, \cdots, c_M)$ 为共享 L2 Cache 分配方案. 因为所有应用都必须被调度运行,所以初始给每个应用分配最少的 Cache 配额,即 C 初值为(1,1,…,1). 算法第 3 行依据待调度应用集合的不同应用的初始微增益 $MMR_i(1)$ 把应用进行排序. 然后进入算法的关键循环. 关键循环每执行一次,就分配一个 Cache 块(第 6 行),分配的对象是得到该 Cache 块性能增益最高的应用(第 5 行). 若 MMR 采用树结构组织,那么第 3 行排序的算法复杂度为 $O(M\log_2 M)$,第 4 行循环的复杂度为 $O(CN - M)$,第 7 行重新对 MMR 排序的复杂度为 $O(\log_2 M)$. 综上所述,算法复杂度为 $O(M\log_2 M + (CN - M)\log_2 M) = O(CN \cdot \log_2 M) = O\left(\frac{C_{L2}}{\Delta}\log_2 M\right)$.

3　实验方法

我们使用全系统模拟器,在吞吐率、失效率、加权加速比和公平性等方面评估 WCP-0、WCP-0.5 和 WCP-1 的性能.

3.1　平台介绍

我们使用基于 Virtutech Simics[9] 的全系统多核模拟器模拟 8 核 CMP 处理器,目标体系结构为 Sparc v9,操作系统为 solaris9. 每个处理器核都有独立的一级指令 Cache 和数据 Cache,片上的 8 个核共享二级 Cache. 一级 Cache 均为 32 KB 两路组相联,二级 Cache 大小为 1MB、16 路组相联. Cache 块大小为 64B,替换策略为 LRU. 表 1 给出目标机的性能参数.

表 1　目标机参数

参数	指标
System	1 Chip, 8 cores per chip
Processor Technology	2 GHz, 0.5ns cycle time
Core Configuration	Single-issue, in-order, no-branch-predictor
L1 D/I Cache	32 KB, 2-way, LRU, 64B Cache Line, 1 cycles
L2 Cache	1MB, 16-way shared, 4 banked, 6 cycles
CC Protocol	Snoop based MESI protocol
Maim Memory	200 cycles

3.2　度量指标

目前有多种度量多道程序类应用系统性能的评估指标. 在此,我们只关注最常用的 4 类评估指标:吞吐率、失效率、加权加速比[4] 和公平性[10]. 假设 IPC_i 为多道程序测试用例的应用 i 的 IPC, MR_i 为应用 i 的失效率, $ExIPC_i$ 为应用 i 独占使用共享 Cache 时的 IPC, $ExMR_i$ 为应用 i 独占使用共享 Cache 时的失效率. 那么,对于由 N 个多线程应用组成的多道程序测试用例,评估函数定义如下:

吞吐率:

$$IPC_{sum} = \sum_{i=1...N} IPC_i.$$

失效率:

$$MR_{sum} = \sum_{i=1...N} MR_i.$$

加权加速比:

$$WeightedSpeedup = \sum_{i=1...N} (IPC_i / ExIPC_i).$$

基于 IPC 的公平性:

$$Fairness_{ipc} = N \Big/ \left(\sum_{i=1...N} (ExIPC_i / IPC_i) \right).$$

基于失效率的公平性:

$$Fairness_{mr} = N \Big/ \left(\sum_{i=1...N} (MR_i / ExMR_i) \right).$$

吞吐率反映的是系统的整体性能,但是较高的系统吞吐率的获得可能以降低 IPC 较低的应用为代价;失效率是从失效率的角度反映系统的整体性能,较低的系统失效率的获得可能以倾向于牺牲失效率高的应用的性能为代价,而往往失效率低并不代表 IPC 高;加权加速比反映的是执行时间的降低情况;公平性以 IPC 性能和失效率为尺度评估系统公平性指标.

3.3　测试用例

为评估 WCP,我们选择科学和工程计算应用组成多道程序类测试用例. 这些应用来自 SPLASH2[11] 测试用例集合,其功能描述和数据输入如表 2 所示.

表 2 Splash2 测试用例功能描述和输入说明

名称	功能描述	数据输入
Fft	快速傅立叶变换	1M 复数
Lu-contiguous	稠密矩阵分解(优化数据存储)	1024 ×1024
Lu-noncontiguous	稠密矩阵分解(数据块不连续存放,非优化)	1024 ×1024
Radix	并行 Radix 排序	10M 整数
Cholesky	稀疏矩阵分解	tk15.O
Ocean-contiguous	海洋模拟(优化数据存储)	258 ×258
Ocean-noncontiguous	海洋模拟(数据块不连续存放,非优化)	258 ×258
Barnes	N-body 模拟	16K particles
Raytrace	三维场景构建	Ball
Radiosity	场景中光效果计算	Room
Water-nsquared	水分子作用力建模	512

基于 Splash2 测试用例,我们构建了如表 3 所示的多道程序测试用例集合.我们构造了 3 类多道程序测试用例,每一类测试用例都需要 8 个线程执行.3 类多道程序的组合模式为:224、1124 和 11222. 224 组合模式的多道程序测试用例集合由 3 个应用组成,分别需要 2 个、2 个和 4 个线程执行;同理 1124 由 4 个应用组成,其中有 2 个应用是单线程、1个是双线程、1个是四线程;11222 由5个应用组成,其中有 2 个是单线程、3 个是双线程.测试用例名称的数字后缀是该测试用例的线程数.Lu_c、lu_n、ocean_c、ocean_n 和 water_n 分别为 lu-contiguous、lu-noncontiguous、ocean-contiguous、ocean-noncontiguous 和 water-nsquared 的简称.

表 3 多道程序测试用例集合

类型	序号	测试用例
224	1	barnes_2 , cholesky_2 , lu_c_4
	2	barnes_2 , fft_2 , ocean_c_4
	3	barnes_2 , ocean_n_2 , ocean_c_4
	4	fft_2 , lu_c_2 , radiosity_4
	5	lu_n_2 , ocean_n_2 , radix_4
	6	ocean_c_2 , radiosity_2 , cholesky_4
	7	ocean_c_2 , readix_2 , radiosity_4
	8	raytrace_2 , water_n_2 , radix_4
1124	9	fft_1 , lu_n_1 , ocean_n_2 , cholesky_4
	10	lu_c_1 , ocean_c_1 , barnes_2 , ocean_n_4
	11	lu_n_1 , radiosity_1 , radix_2 , ocean_c_4
	12	ocean_c_1 , radiosity_1 , barnes_2 , lu_c_4
	13	ocean_n_1 , radiosity_1 , lu_c_2 , cholesky_4
	14	radix_1 , raytrace_1 , fft_2 , lu_n_4
	15	raytrace_1 , lu_n_1 , water_n_2 , barnes_4
	16	barnes_1 , fft_1 , lu_n_2 , ocean_n_4
11222	17	barnes_1 , ocean_n_1 ,barnes_2 , lu_n_2 , radix_2
	18	cholesky_1 , lu_n_1 , barnes_2 , radiosity_2 , water_n_2
	19	fft_1 , radiosity_1 , cholesky_2 , ocean_c_2 , raytrace_2
	20	fft_1 ,ocean_n_1 , cholesky_2 ,lu_c_2 , radiosity_2
	21	ocean_n_1 , radix_1 , fft_2 , lun_n_2 , ocean_c_2
	22	ocean_c_1 , raytrace_1 , fft_2 , ocean_n_2 , raytrace_2
	23	ocean_n_1 , water_n_1 , cholesky_2 , lu_c_2 , radiosity_2
	24	radix_1 , ocean_c_1 , barnes_2 , fft_2 , ocean_n_2

每一组多道程序测试用例都执行 3 亿条指令,每隔 1000 万条指令执行一次 Cache 划分.其中前一亿条指令用来初始化各级 Cache 的初始状态,后两亿条指令用来统计 Cache 划分算法性能.对于 8 核系统,模拟器大概执行 24 亿条指令.应用在 AMRM 中分配的伪 Cache Set 的数目为该应用线程数的 32 倍.

4 实验结果和分析

我们通过比较 WCP-0、WCP-0.5 和 WCP-1 在各种度量指标下的差异来评估 WCP.其中 WCP-0 是传统的失效率最低的动态共享 Cache 划分算法;WCP-0.5 和 WCP-1 是权值为非 1 的两种动态共享 Cache 划分算法.我们没有比较多个进程共享使用 Cache 的 LRU 替换策略,因为文献[4-5]的工作已经把 LRU 策略和 Cache 划分机制的性能差异做了详尽的比较.

4.1 吞吐率

图 4 比较了 3 种 Cache 划分算法的吞吐率差异.标号 1 到标号 24 对应多道程序测试用例 1 到 24,25 是 24 个多道程序测试用例集吞吐率的平均值.24 个测试用例中,有 2 个测试用例 3 种 Cache 划分算法具有相同的吞吐率性能,此时 WCP-0、WCP-0.5 和 WCP-1 是等效的.有 3 个测试用例 WCP-0 的性能优于 WCP-0.5 和 WCP-1,可见加权 Cache 划分算法有些情况下性能低于传统 Cache 划分算法.对于其余的 19 个测试用例,WCP-0.5 和 WCP-1 都高于或等于 WCP-0,可见对于多线程应用,加权 Cache 划分算法通常会获得更优的性能.有 8 个测试用例 WCP-0.5 和 WCP-1 具有相同的吞吐率,在其余的 16 个测试用例中,WCP-1 性能较优的有 11 个,而 WCP-0.5 性能较优的有 5 个.说明对于加速比较好的科学计算程序,WCP-1 获得更优性能的可能性更大.

在所有的测试用例中,WCP-1 的平均吞吐率比 WCP-0 高 5.5%,比 WCP-0.5 高 2.2%;WCP-0.5 平均吞吐率比 WCP-0 高出 2.9%.说明 Cache 划分

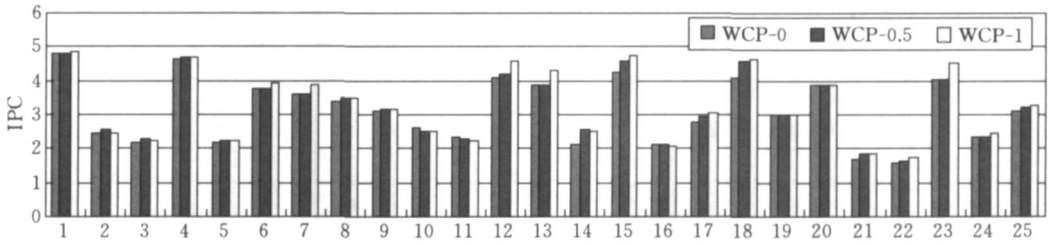

图 4　吞吐率测试结果

算法的选择对系统吞吐率的影响很大,因此通过 Cache 划分支持工具集设置 Cache 划分算法是必要的.

4.2　失效率

图 5 比较了 3 种 Cache 划分算法的失效率差异. 标号 1 到标号 24 对应多道程序测试用例 1 到 24, 25 是 24 个多道程序测试用例集失效率的平均值. 在所有的 24 个测试用例中,WCP-0 的失效率均为最低的. WCP-1 的平均失效率比 WCP-0 高 2.3%,比 WCP-0.5 高 1.7%;WCP-0.5 平均失效率比 WCP-0 高出 0.6%,说明 WCP-1 的高性能是以更高的系统失效率为代价的. 综合 4.1 和 4.2 可得:具有较低失效率的 Cache 划分方案不一定获得较高的 IPC 吞吐率.

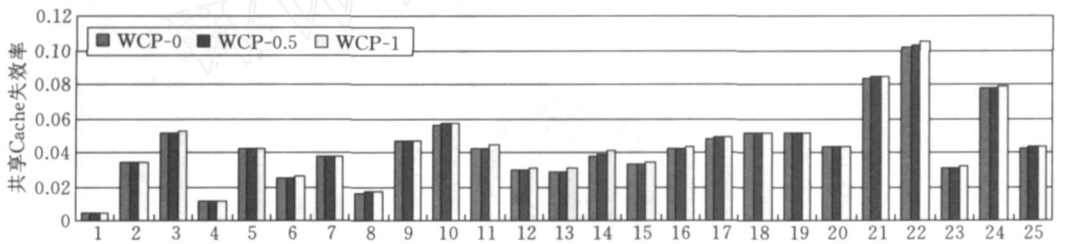

图 5　失效率测试结果

传统的共享 Cache 划分算法 WCP-0 倾向于把共享 Cache 划分给获得的共享 Cache 失效率最多的应用,以降低共享 Cache 的平均失效率. 因此,WCP-0 具有最低的平均失效率. 然而,系统的吞吐率等其它性能评估标准并不只是受限于 L2 Cache (当 L2 Cache 共享)的失效率. 例如,系统的吞吐率,当系统主频确定时,主要受限于各个应用的 L1 Cache 和 L2 Cache 的失效率和失效开销. 而 WCP-0.5 和 WCP-1 则倾向于把共享 Cache 划分给并行度高的应用程序,以开发系统的潜在并行度,因此具有更高的吞吐率.

4.3　加权加速比

图 6 比较了 3 种 Cache 划分算法的加权加速比. 标号 1 到标号 24 对应多道程序测试用例 1 到 24, 25 是 24 个多道程序测试用例集加权加速比的平均值. WCP-1 的平均加权加速比比 WCP-0 高 1.2%,比 WCP-0.5 高 0.1%;WCP-0.5 平均加权加速比比 WCP-0 高出 1.1%. 加权加速比反映了应用平均执行时间的减少情况,WCP-0.5 和 WCP-1 平均加速比高于 WCP-0 说明加权 Cache 划分算法有利于降低应用的平均执行时间.

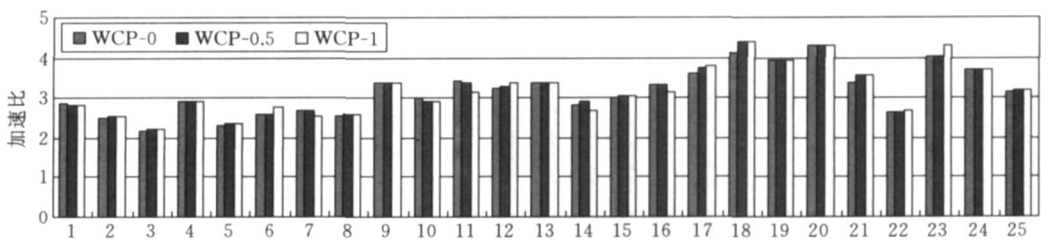

图 6　加权加速比测试结果

4.4　公平性

Cache 划分算法常常会以降低某个应用的性能为代价提高系统的整体性能. 我们分别采用基于失效率和 IPC 的调和平均数作为公平性的评估标准. 基于失效率的公平性测试结果如图 7 所示,从图中可见,不同 Cache 划分算法对相同应用的公平度波

动较大. 但平均而言,WCP-1 的公平性比 WCP-0 高出 6.4%,比 WCP-0.5 高出 9.4%;而 WCP-0.5 的公平性比 WCP-0 低 2.8%. 说明 WCP-0 虽然具有最低的失效率,但是公平性较低. 基于 IPC 的公平性测试结果如图 8 所示. 由图可见,不同 Cache 划分算法对相同应用的公平度波动较小. 平均而言,WCP-0、WCP-0.5 和 WCP-1 具有类似的公平性. 综合图 7,8 可得,WCP-1 具有相对较高的公平性.

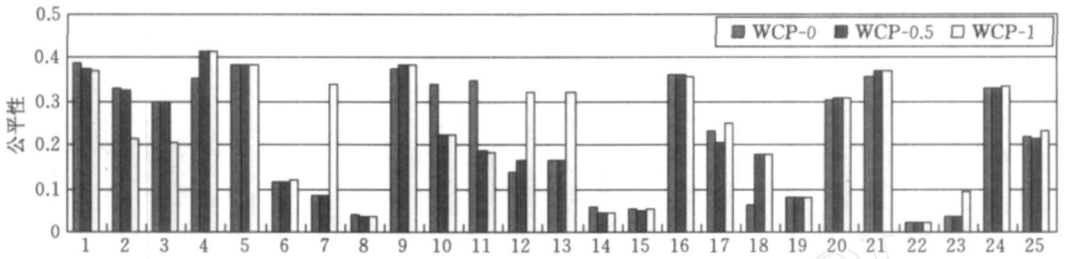

图 7 基于失效率的公平性测试结果

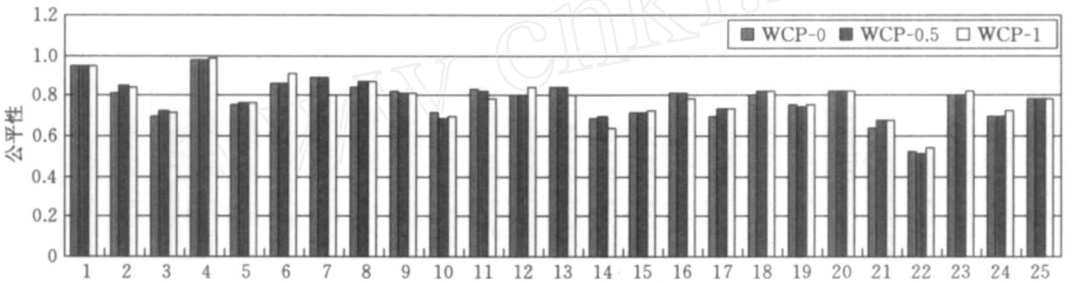

图 8 基于 IPC 的公平性测试结果

4.5 硬件开销

WCP 的硬件开销主要来自于 AMRM,对于拥有 8 核的多核处理器,若每个应用最少需要 32 个伪 Cache Set,则需要伪 Cache Set 的数目是 32 ×8 = 256. 若地址空间为 32 位,Cache 块大小为 64 字节,则 Tag 的长度为 26 位;若共享 Cache 的相联度为 16,则 LRU 标志位的长度为 4;此外每个 Cache 块还需 2b 标志位:Dirty 和 Invalid. 则每个伪 Cache 块的存储开销为(26 + 4 + 2) ×16 = 512b = 64B. 此外,若计数器的长度为 4B,则计数器阵列的存储开销为 4 ×17 ×8 = 544B. 综上所述,AMRM 的存储开销为 64 ×256 + 544 = 16.54 KB. 16 路组相联的 1MB 共享 Cache 的存储开销为 64 KB + 1024 KB = 1086 KB. 若不考虑加法器等功能部件的存储开销,AMRM 所需的存储空间为共享 Cache 的 1.54%. 可见 WCP 的硬件开销很低.

此外,基于列划分的多核处理器,每个核还需要一个比特向量标示属于自己的 Cache 列,其存储开销为 16 ×8 = 144b = 16B,相比 AMRM 可忽略不计.

5 结 论

本文研究了多核处理器共享 Cache 的划分问题,我们面向多道多线程应用提出加权 Cache 划分框架 ——WCP. 包括:基于应用的失效率监控器、加权共享 Cache 划分算法和该框架的模拟实现和评测结论. 实验结果表明,加权 Cache 划分算法虽然具有更高的失效率,但具有较高的 IPC 吞吐率、加权加速比和公平性.

参 考 文 献

[1] Kalla R, Balaram S et al. IBM Power 5 chip: A dual-core multithreaded processor. IEEE Micro, 2004, 24(2): 40-47

[2] Kongetira P, Aingaran K et al. Niagara: A 32-way multithreaded Sparc processor. IEEE Micro, 2005, 25(2): 21-29

[3] Kim S, Chandra D, Solihin Y. Fair Cache sharing and partitioning in a chip multiprocessor architecture// Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques. Orlando, Florida, 2004: 111-122

[4] Qureshi M K, Patt Y N. Utility-based Cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches// Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. Antibes Juan-les-Pins, France, 2006: 423-432

[5] Suh G E, Rudolph L, Devadas S. Dynamic partitioning of shared Cache memory. Journal of Supercomputing, 2004, 28

(1): 7-26

[6] Iyer R. CQoS: A framework for enabling QoS in shared caches of CMP platforms/ / Proceedings of the 18th Annual International Conference on Supercomputing. Malo, France 2004: 257-266

[7] Iyer R, Zhao L, Guo F et al. QoS policies and architecture for Cache/ memory in CMP platforms. SIGMETRICS Performance Evaluation Review, 2007, 35(1): 25-36

[8] Chiou D, Jain P, Rudolph L et al. Application-specific memory management for embedded systems using software-controlled caches/ / Proceedings of the 37th Conference on Design Automation. Los Angeles, California, United States: 2000: 416-419

[9] Magnusson P S, Christensson M, Eskilson J et al. Simics: A full system simulation platform. Computer, 2002, 35(2): 50-58

[10] Luo K, Gummaraju J, Franklin M. Balancing throughput and fairness in SMT processors/ / Proceedings of the 21st International Symposium on Performance Analysis of Systems and Software. Tucson, AZ, 2001: 164-171

[11] Woo S C, Ohara M, Torrie E et al. The SPLASH-2 programs: Characterization and methodological considerations/ / Proceedings of the 22nd Annual International Symposium on Computer Architecture. S. Margherita Ligure, Italy 1995: 24-36

SUO Guang, born in 1980, Ph. D. candidate. His research interests focus on multi-core processor architecture and parallel computing.

YANG Xue-Jun, born in 1963, professor, Ph. D. supervisor. His main research interests include parallel computer architecture, parallel computing, advanced compiler, fault-tolerant computing and operating system.