基于分布式合作 cache 的私有 cache 划分方法

李 浩, 谢伦国
(国防科学技术大学 计算机学院, 长沙 410073)

摘 要: 当片上多处理器系统上运行多个不同程序时,如何给这些不同的应用程序分配适当的 cache 空间成为一个难题。Cache 划分就是解决这一难题的有效方法,目前大部分的划分方法都是针对最后一级共享 cache 设计的。私有 cache 划分(private cache partitioning,PCP)方法采用一个分布式一致性引擎(DCE)把多个私有 cache 组织在一起,最后通过硬件信息提取单元获得多个程序在不同 cache 路上的命中分布情况,用于指导划分算法的执行,最后由每个 DCE 根据划分算法运行的结果对 cache 空间进行划分。实验结果表明 PCP 方法降低了失效率,提高了程序执行性能。

关键词: 片上多处理器; 分布式合作 cache; 共享设计; 私有设计; 私有 cache 划分
中图分类号: TP399 **文献标志码**: A **文章编号**: 1001-3695(2012)01-0229-05
doi:10.3969/j.issn.1001-3695.2012.01.064

Private cache partitioning based on distributed cooperative cache in chip-multiprocessors

LI Hao, XIE Lun-guo
(*School of Computer Science*, *National University of Defense Technology*, *Changsha* 410073, *China*)

Abstract: When there are several application running on chip-multiprocessors(CMPs), it is a problem to allocate the on-chip cache capacities between these competing applications. Cache partitioning is commonly used to solve this problem. Existing cache partitioning schemes either dedicate to the shared design or partition the last level cache depending on limited memory information. This paper presented private cache partitioning, a low-overhead, runtime mechanism that partitioned all of the private low level caches which were organized as a large shared cache by a distributed directory. The experiment results show that PCP reduces the overall missrate of competing applications and improves the throughput as well as the weighted speedup.

Key words: chip-multiprocessor; distributed cooperative cache; shared design; private design; private cache partitioning

0 引言

最近几年来,随着半导体制造工艺的飞速发展,单个处理器核的主频已经逐渐逼近极限。要想使处理器的运算速度继续沿着摩尔定律高速发展,光靠提升单个处理器核的主频已经无法达到目的。因此人们将多个处理器核集成在一个芯片上形成片上多处理器(chip multi-processor,CMP)系统。英特尔公司最新的桌面处理器 i7-990X 已经在单个芯片上集成了 6 个处理器核[1],而该公司最新的针对高端服务器开发的至强处理器(Xeon)7500 系列则在单个芯片上集成了 8 个处理器核[2]。AMD 公司针对服务器开发的皓龙(Opteron)6000 系列则在单芯片上集成了 12 个处理器核[3,4]。Sun 公司最新的高端处理器 UltraSPARC T3(Niagara-3)更是将单芯片上的处理器核数增至 16 个[5]。片上多处理器系统已经成为各大微处理器制造商发展的主流。随着技术的发展,片上处理器核的数目越来越大,处理器核数目增加的速度远远超过了芯片管脚数目的增加速度,这使得片外访存更加成为处理器性能的瓶颈。为了降低片外访存的不良影响,人们通常是在片上放置大容量的 cache[1~5] 来减少处理器对片外存储器的访问需求。

与传统单核处理器相比,片上多核处理器在处理多任务多线程程序时具有无可比拟的优势。但是,当多个线程运行在 CMP 系统上时,它们对片上硬件资源特别是 cache 的使用会造成冲突,从而降低处理器的性能。Cache 划分技术[6~11] 的提出就是为了解决这个问题。目前基于共享 cache 设计的 cache 划分技术有 cache 静态最优划分[6]、动态划分[7]、基于利用率 cache 划分(utility-based cache partitioning,UCP)[8]、公平 cache 划分[9]等;基于私有 cache 设计的 cache 划分技术有自适应私有/共享 cache(adaptive shared private NUCA,ASP-NUCA)[10]、弹性合作 cache(elastic cooperative caching,ECC)[11]等。当采用最后一级私有 cache 作为划分的目标时,因为私有 cache 相对于共享 cache 具有更大片外访存需求,所以首先要解决的是如何降低私有 cache 设计的片外访存次数。ASP-NUCA 和 ECC 的做法是从各个处理器核自己私有 cache 中划分出一块提供给其他的处理器核共享使用,而它们划分算法的最终结果就是各自私有 cache 中共享部分的大小,这是一种粗粒度的划分机制,不能够根据每个线程对 cache 空间的使用效率进行精确的划分。本文提出一种针对最后一级 cache(本文中最后一级 cache 为 L2 cache)的私有 cache 划分(private cache partitioning,

收稿日期: 2011-06-08; **修回日期**: 2011-07-11
作者简介:李浩(1980-),男,湖南岳阳人,博士研究生,主要研究方向为高性能计算机体系机构(li_haozi@163.com);谢伦国(1947-),男,湖南隆回人,教授,博导,主要研究方向为高性能计算机、微处理器体系结构.

PCP)方法。本文首先通过一种分布式一致性引擎(distributed coherence engine,DCE)[12]将多个私有 cache 组织成一个逻辑上共享的 cache,然后在这个逻辑共享 cache 之上采用类似基于共享设计的 cache 划分方法进行精细划分。实验结果表明,本文提出的私有 cache 划分方法降低了多个程序总的 cache 失效率,提高了多程序的最终执行性能。

1　研究背景

1.1　私有和共享设计

CMP 的最后一级 cache 有私有和共享两种基本组织方式。共享设计把片上所有的最后一级 cache 存储体连接在一起,统一编址组成一个供所有处理器核共享的大容量底层 cache,每个处理器根据地址索引到相应的 cache 存储体中请求数据。处理器核对最后一级 cache 的访问速度取决于请求处理器核跟数据所在 cache 存储体之间的距离以及网络的拥塞情况。为了保持一致性,每个数据块都增加了一些目录位来跟踪都有哪些处理器拥有本数据块的副本,这样在数据块被替换或更改时可以有针对性地发送作废信号。

在私有设计中,每个 cache 存储体都被看做是本地处理器核私有的大容量 cache 以提高命中访问速度。为了保证数据的一致性,需要设置一个高带宽的目录机制。这个目录机制其实就是 cache 中 tag 阵列的复制,且根据索引分布在 CMP 的各个存储体中。为了减少片外访存,私有设计允许处理器核访问相邻处理器核对应的 cache 存储体,并能够通过 cache 存储体之间的相互传输来获得数据;但是这种传输需要请求节点、目录节点以及数据块拥有节点三者之间的通信,因此带来的消耗较大。已有的研究[11]表明,对大多数应用来说,私有设计的性能要比共享设计更优。

1.2　集中式目录和分布式目录

对于基于私有设计的大容量底层 cache 来说,如何保证数据的一致性成为一个问题,传统的一致性协议主要有监听协议和目录协议。监听协议实现简单但是不适用于处理器核很多的 CMP cache 系统,因此未来的 CMPcache 主要采用目录协议来保持数据一致性。现有的目录机制根据目录所处的位置可以分为集中式和分布式目录结构。例如,合作式 cache(CC)[13]采用了一个集中式的一致性引擎(central coherence engine,CCE)来实现多个 cache 存储体之间数据块共享和 cache 空间的共享。这种机制将每个 L2 cache 的 tag 阵列的复制集中在一起形成一个集中式的目录,如图 1 所示。在本地 L2 cache 发生失效或者是数据块被驱逐出 L2 cache 时,请求都会发送给集中式目录。采用高带宽的集中式目录可以使各个私有 L2 cache 实现快速的数据共享和 cache 空间的共享,但是当处理器数目进一步增加时,集中式目录的访问带宽将会成为一个性能瓶颈,因此 Herrero 等人[12]提出一种分布式一致性引擎。在这种机制中,目录跟各个处理器核一样分布在芯片的各个位置,如图 2 所示。当发生本地 L2 cache 失效或者是数据块驱逐时,请求根据数据块的地址发送给某一个节点中的目录单元。分布式目录相比集中式目录具有更好的可扩展性。

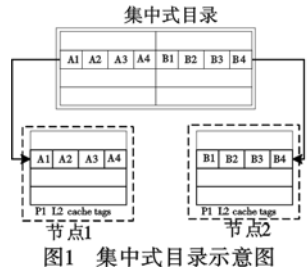

图1　集中式目录示意图

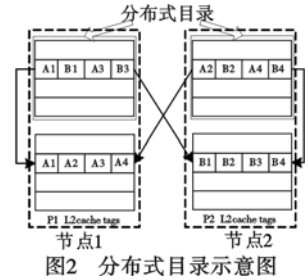

图2　分布式目录示意图

1.3　分布式合作 cache

本文划分的基础是分布式合作 cache(distributed cooperative cache,DCC)[12]。DCC 是一种采用 DCE 作为一致性协议的私有 cache 机制。图 3 为分布式合作 cache 的基本结构示意图。在 DCC 中,用于维持数据一致性的目录被划分成多个 DCE,当发生本地 L2 cache 失效时,访存请求会根据请求的地址发送至相应的 DCE。如果 DCE 中有该数据的标志(tag)存在(DCE 命中),则说明数据块在其他的 L2 cache 中存在,这时请求就会转发至数据块拥有者处。如果请求在 DCE 处发生失效,则由 DCE 向主存发送请求,主存数据返回后,首先会发送给请求处理器核的本地 L2 cache,同时在 DCE 中保留相应的 tag。DCE 中 tag 的替换策略为 LRU 策略。如果 DCE 中的某一个 tag 项被替换出去时,相应的私有 L2 cache 中的数据块同样离开芯片。当一个数据块的 tag 在 DCE 中,但是数据块在私有 L2 cache 中被替换出来时,数据块并不马上离开芯片,而是采用 N-机会推进(N-chance forwording)[12,13]的方法来向其他 L2 cache 进行转移。DCE 的相联度独立于私有 cache 的相联度,两者可以相同,也可以不同。

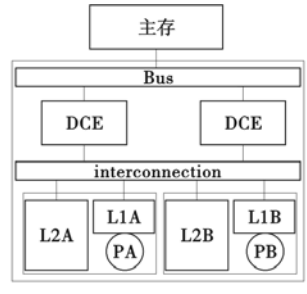

图3　分布式合作cache基本结构示意图

分布式合作 cache 中 DCE 的工作机制其实跟分布式共享 L2 cache[14,15]的工作机制类似,不同的是在分布式共享 L2 cache 中,数据块和 tag 是绑定在一起的;而在 DCE 中,只包含了 tag,而数据块则放置在各个私有 L2 cache 中。通过这种方式,分布式合作 cache 既获得了跟私有 L2 cache 一样的命中延迟,又获得了跟共享 L2 cache 一样的 cache 空间。假设在一个 4 核 CMP 系统中,每个私有 L2 cache 都是 4 路(way)组相联,组(set)数为 8 组,DCE 的相联度为 8,每个处理器核对应的本地 DCE 的组数也为 8,则这个 4 核 CMP 系统的 L2 cache 从逻辑上来看,类似于实现了一个组数为 32 组的 8 路组相联共享 cache。

当多个线程运行在这样的一个逻辑上共享的分布式合作 cache 上时,会遭遇在共享 cache 上运行时一样的问题,多个线程会竞争 DCE 上的存储空间。本文提出的私有 cache 划分(PCP)方法,在分布式合作 cache 的基础上,对多个程序在共享 DCE 空间上进行划分,缓解因多线程冲突而带来的 cache 失效率增加的问题。

2　私有 cache 划分

2.1　划分的硬件框架

本文提出的私有 cache 划分机制框架结构如图 4 所示。每个处理器核都有独立的 L1 指令和数据 cache,还有一个私有的 L2 cache;每个处理器核上运行一个应用程序。每个私有 L2 cache 都对应一个 DCE,所有的私有 L2 cache 通过众多 DCE 组织在一起,形成一个逻辑上的共享 cache。私有 cache 划分机制主要由两部分组成,即应用程序特征提取单元(profiler)和私有 cache 划分算法单元。Profiler 主要负责在每个划分区间内对处理器核上正在运行的应用程序进行 cache 命中分布信息的提取,用于指导划分算法单元的执行。而 cache 划分算法单元在每个划分区间结束时根据 profiler 单元获得的信息运行划分算法,并将划分算法的结果发送给各个 DCE 用于实现指导划分的执行。

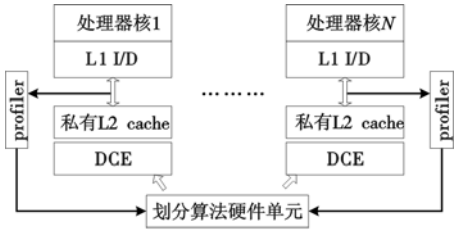

图4　私有cache划分框架

2.2　应用程序特征提取单元

为了能够知道应用程序在占用不同数目 cache 路时的失效率变化情况,给每个处理器核增加一个额外 tag 目录(auxiliary tag directory,ATD),就像给每个处理器核分配了一个虚拟的私有 L2 cache。ATD 跟 DCE 的组相联结构相同,采用 LRU 替换策略。每个 ATD 仅仅包含目录项,不包含数据块。当 L1 发生失效时,请求会同时发向实体 L2 cache 和对应处理器核的 ATD。ATD 和真正的私有 L2 cache 一样运转,只不过它不会返回数据,而是用来统计处理器核在不同路上的命中信息。由于基本的 LRU 替换策略遵循一种堆栈机制,即如果一个数据在 cache 路数为 N 时命中,那么它在 cache 路数大于 N 时也必然命中,因此根据应用程序在不同 cache 路上的命中次数分布情况,可以得到应用程序占用 cache 路数目发生变化时失效率的变化情况。假设一个 8 路组相联的 ATD,设置 8 个计数器分别为 count(1),count(2),…,count(8),分别统计从 ATD 的 MRU 位置到 LRU 位置各路总的命中次数,则该应用程序占用的 cache 路数从 a 增加到 b 时失效次数减少量 U_a^b 可以计算为

$$U_a^b = \mathrm{count}(b-a+1) + \mathrm{count}(b-a+2) + \cdots + \mathrm{count}(b) \qquad (1)$$

因为每一个处理器核都要有一个对应的 ATD,每个 ATD 的大小都跟真正的共享 L2 cache 目录一样大,当处理器核数目增加时,ATD 的硬件开销将会变得不可接受。采用组抽样技术[12]来减少硬件开销。ATD 以某一个固定间隔选取组(该策略叫做简单静态策略)来进行统计,最后得到的数据近似全局的统计数据。Herrero 等人[12]认为,总共选取 32 组作为抽样进行统计可以很好地近似全局统计数据。

2.3　PCP 划分算法

在每个划分区间结束时,应用程序特征提取单元可以获得每个应用程序在各个 cache 路上的命中次数分布情况。根据式(1)可以很容易地得到各个应用程序在占用的 cache 路数从 a 增加到 b 时失效次数减少量 U_a^b。假设 K 个应用程序共用一个 N 路的 cache,$C(i)$ 代表划分给应用程序 i 的 cache 路数,$U(i)$ 代表应用程序 i 的失效次数减少量,则 PCP 划分可以归结为求解下面的规划问题:

$$\begin{cases} \sum_{i=1}^{K} C(i) = N \\ \text{maximize} \sum_{i=1}^{K} U(i)_1^{C(i)} \end{cases} \qquad (2)$$

满足式(2)的向量($C(1)$,$C(2)$,…,$C(K)$)即为所求的目标划分。当应用程序只有两个时,N 路 cache 的划分组合只有 $N+1$ 种,因此式(2)比较容易求解。但是当应用程序数量增加时,划分组合的数量会呈指数值增长,如当有 4 个应用程序共享一个 32 路组相连 cache 时,划分组合有 6 545 种,当应用程序数目增加到 8 个时,划分组合猛增至 15 380 937 种。式(2)的求解问题已经被证明是一个 NP 难题(NP-hard)[8]。本文采用超前算法[8]来解决这个问题。超前算法的时间复杂度为 $O(N^2)$,对于一个 32 路组相联 DCE,采用超前算法只需要 512 次操作。在本文的研究中,每一个应用都至少要分配一路,并且在每个划分区间结束开始一个新的划分区间时,将 ATD 中所有命中计数器的值减半,使得 ATD 得到的命中信息具有局部时间相关特征[8]。

2.4　DCE 中替换算法的改变

为了在 DCE 上实现 cache 划分策略,本文对 DCE 上的 LRU 算法(或者是伪 LRU 算法)进行扩展。给 DCE 中每一个 cache 块的 tag 中增加 $\log_2 N$(N 为处理器核数目)位用于标志该块属于哪个处理器核(最先请求该块的处理器核)。每个划分区间结束时,通过 2.3 节的算法,可以得到每个处理器核所分配的最高占用 cache 块数值 $C(i)$,用于指导下一个划分区间内的替换策略。在每一个划分区间内,当一个 cache 失效,DCE 替换引擎统计引发该失效的应用程序 i 在 cache 中占用的路数,如果不超过 $C(i)$ 值,则在本应用占用的 cache 块之外选择一个 LRU 块驱逐,如果达到或者是超过了 $C(i)$ 值,则在本应用占用的 cache 块中选择一个 LRU 块驱逐。在本文中,每执行 5 M 指令运行一次划分算法,$C(i)$ 值刷新一次。

3　测试与分析

为了全方位地评价 PCP 方法,本文在全系统模拟器上分别实现了私有 L2 cache(L2P)、分布式合作 cache[12]以及 PCP 机制,从 L2 cache 失效率、系统吞吐率(IPC)、加权加速比等多个方面对它们进行了比较。

3.1　测试平台

GEMS 模拟器[16]是威斯康辛大学在 Virtutech 公司开发的通用并行模拟器 Simics[17]的基础上扩展实现的。Simics 是一个全系统虚拟机,可以进行功能模拟和时序模拟,它支持三种模拟模式:正常模式、微体系结构模式和暂停(stall)模式。GEMS 模拟器在暂停模式下增加了对 cache 系统和互连网络的模拟。本文采用 GEMS 模拟器对几种机制进行模拟测试。

本文模拟实验平台为一个 4 核 CMP 系统,每个处理器核为乱序执行的超标量处理器,有自己的 L1 指令和数据 cache,以及一个私有的 L2 cache。具体参数如表 1 所示。

表 1　模拟实验平台基本配置

处理器核	4 个,4 流出,乱序执行;指令窗口大小:64;保留站大小:128
L1 cache	Icache;Dcache:32 KB,数据块大小 64 Byte,4 路组相联
L2 cache	1 MB,数据块大小为 64 Byte,4 路组相联,本地命中访问延迟为 15 个时钟周期,LRU 替换策略
DCE	8/16 路组相
互连网络	2×2 Mesh,点对点通信延迟 15 个时钟周期
主存	150 个时钟周期

3.2　度量指标

多道程序并行执行时的度量指标有很多种,本文选取其中的三种:吞吐率、L2 cache 失效率以及加权加速比。设 IPC_i 为应用程序 i 在并行执行时的 IPC,$singleIPC_i$ 为应用程序 i 单独执行在私有 L2 cache(没有通过目录实现共享 L2 cache)上时的 IPC,$missRate_i$ 为应用程序 i 在多程序同时执行时的 L2 cache 失效率,则这三种度量指标定义为

吞吐率 $IPC_{sum} = \sum IPC_i$

失效率 $missSum = \sum missRate_i$

加权加速比 $WeightedSpeedup = \sum (IPC_i / singleIPC_i)$

这三者中,吞吐率反映了处理器单位时间里的处理能力,失效率反映了系统的整体性能,而加权加速比直接反映了执行时间的变化。

3.3　测试程序

为了评估 PCP,选择科学和工程计算应用组成多道程序测试用例。这些应用来自 SPLASH2 测试程序集合,这些程序的基本访存特点如表 2 所示。

表 2　SPLASH2 测试程序参数

code	problem size	total instr/M	total reads/M	total writes/M	shared reads/M	shared writes/M
Ocean	258×258	379.93	81.89	18.93	80.26	17.27
Radiosity	room	2 832.47	499.72	284.61	261.08	21.99
Water-Sp	512 molecules	435.42	72.31	32.73	60.54	22.64
Barnes	16K particles	2 002.79	406.85	313.29	225.05	93.23
fft	64K points	34.79	4.07	2.88	4.05	2.87
Cholesky	TK15.O	539.17	111.86	28.03	75.87	23.31
Radix	1M integers, radix1024	50.99	12.06	7.03	12.06	7.03

基于 SPLASH2 测试用例,构建了如表 3 所示的多道程序测试用例集合。构造了两类多程序测试用例集合,每一类测试用例都需要四个线程同时执行。这两类测试程序组合模式分别为 22 和 112。22 组合模式的多程序测试用例由两个应用程序组成,每个应用程序都是两个线程;而 112 组合模式的测试用例由三个应用程序组成,前两个应用程序为单线程执行,而最后一个是双线程执行。

本文采用 Simics 提供的 magic 指令控制程序的执行,在每个程序完成初始化 cache 的过程后,每个程序最少执行 250 M 条指令(如果有的程序先执行完就循环执行以等待其他程序也执行完 250 M 指令),每 5 M 时钟周期为一个划分区间。

3.4　实验结果和分析

本文通过比较私有 L2 cache(L2P)机制、DCC 机制(8 路)、PCP 机制(8 路)以及 16 路的 DCC(图中标志 DCC2)和 16 路 PCP 机制(图中标志 PCP2)在吞吐率、失效率以及加权加速比三个性能度量指标上的变化来评估 PCP 方法的性能。

表 3　多程序测试用例集合

类型	序号	测试用例	类型	序号	测试用例
22	1	Barnes, Cholesky	112	9	Barnes, Cholesky, fft
	2	Barnes, fft		10	Barnes, fft, Ocean
	3	Barnes, Ocean		11	Barnes, Ocean, Radiosity
	4	Radiosity, Ocean		12	Radiosity, Ocean, Water-Sp
	5	Water-Sp, Radix		13	Water-Sp, Radix, Cholesky
	6	Radix, Cholesky		14	Radix, Cholesky, Radiosity
	7	Radiosity, Radix		15	Radiosity, Radix, fft
	8	Water-sp, fft		16	Water-Sp, fft, Radiosity

1)吞吐率

图 5 比较了五种机制在吞吐率的差异,标号 1~16 分别对应测试用例中的 1~16 组测试程序,标号 17 为所有测试用例吞吐率平均值。对于大部分测试用例来说,PCP 的吞吐率要比 DCC 高,但是对标号 5、标号 10 以及标号 16 来说,PCP 没有起到提高性能的作用。平均来说,8 路组相联的 PCP 机制相比 8 路组相联的 DCC 机制吞吐率提高了 2.3%,当相联度都为 16 路时,提高了 2.4%。

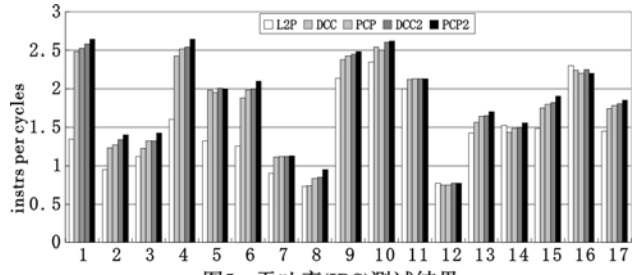

图5　吞吐率(IPC)测试结果

2)失效率

图 6 给出了五种机制在失效率上的测试结果。标号 1~16 分别对应测试用例中的 16 组测试程序,标号 17 为所有 16 组测试用例的平均失效率;Y 轴表示平均每条指令失效次数(misses per instruction,MPI)。从图 6 可以看出,对于大部分测试用例来说,PCP 的失效率比 DCC 都要小,失效率平均降低了 6.5%(8 路)和 6.6%(16 路)。

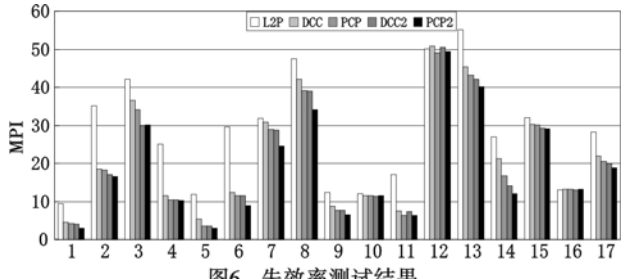

图6　失效率测试结果

3)加权加速比

图 7 给出了五种机制在加权加速比上的测试结果。标号 1~16 分别对应测试用例中的 16 组测试程序,标号 17 为 16 组测试用例的平均加权加速比。加权加速比反映了测试程序在执行时间上的变化。从图 7 来看,除了第 7、10、12、15 道测试用例的加权加速比提升不明显,对于其他的测试用例,PCP 性能都要优于 DCC,其加权加速比平均提高了 2.5%(8 路)和 2.7%(16 路)。

3.5　硬件开销分析

相对于 DCC,PCP 的硬件开销主要来源于两个部分:a)在 DCE 中每一项都增加 $\log_2 N$ 位用于标志该 cache 块属于哪个处

理器核;b)Profiler 模块所用开销。对于一个 4 核 8 路 1 024 组的 DCE 来说,假设组抽样数为 32,则第一部分的开销为

$$(8 \times 1024 \times 2)/8 = 2\ \text{KB}$$

Profiler 模块所用开销中最主要的就是 ATD 所用开销。表 4 详细列出了 ATD 所用开销。Profiler 模块中还包括了一些计数器,但是这些计数器的硬件开销相比 ATD 的开销可以忽略不计。

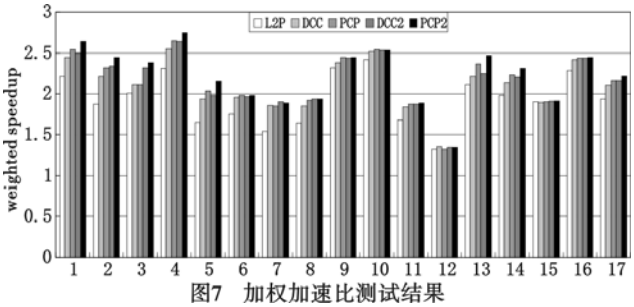

图7　加权加速比测试结果

表 4　ATD 额外硬件开销统计

ATD 目录项(1 有效位 +24 位 tag +4 位 LRU)	29 位
每一组 ATD 项数	8
抽样组数	32
ATD 目录开销(29 位 ×16 路 ×32 组)	928 B

因此 PCP 总的硬件开销不超过(2 KB +928 Byte) ×4 <12 KB。因此对于一个 4 核的 CMP 来说,PCP 的额外硬件开销不超过:

$$12\ \text{KB}/4\ 096\ \text{KB} \approx 0.3\%$$

4　结束语

CMP 的持续发展给片上 cache 系统带来了巨大的压力。本文在分布式合作 cache 的基础上提出了一种私有 cache 划分(PCP)方法。PCP 方法采用一个硬件信息提取单元(profiler)来获得不同程序各自的命中分布情况,用于指导划分算法的执行。实验结果表明,PCP 机制能够在一定程度上减小分布式合作 cache 上运行的竞争程序总的失效率,提高了多个程序的实际执行性能。

由于篇幅问题,本文没有对 PCP 机制在公平性、QoS、优先级支持等方面的表现进行分析,在未来的工作中,笔者将进一步加强这一方面的研究。

参考文献:

[1] Intel. Intel® Core™ i7 processor extreme edition: product brief[R/OL]. (2010-12-30). http://download.intel.com/products/processor/corei7EE/extremeprodbrief.pdf.

[2] Intel. Leading virtualization performance and energy efficiency in a multi-processor server: Xeon 7500 product brief [R/OL]. (2010-12-30). http://download.intel.com/products/processor/xeon/7500prodbrief.pdf.

[3] AMD. AMD Phenom™ II processors product brief [R/OL]. (2010-12-30). http://www.amd.com/us/products/desktop/processors/phenom-ii/Pages/phenom-ii-product-brief.aspx.

[4] AMD. AMD Opteron™ 6000 series platform [R/OL]. (2010-12-30). http://www.amd.com/us/products/server/processors/6000-series-platform/pages/6000-series-platform.aspx.

[5] Sun Microsystems. UltraSPARC T3 processor [R/OL]. (2010-12-30). http://www.oracle.com/us/products/servers-storage/servers/sparc-enterprise/t-series/sparc-t3-171613.html.

[6] STONE H S, TUREK J, WOLF J L. Optimal partitioning of cache memory [J]. IEEE Trans on Computers, 1992, 41(9): 1054-1068.

[7] SUH G E, RUDOLPH L, DEVADAS S. Dynamic partitioning of shared cache memory [J]. Journal of Supercomputing, 2004, 28(1): 7-26.

[8] QURESHI M K, PATT Y N. Utility-based cache partitioning: a low-overhead, high-performance, runtime mechanism to partition shared caches [C]//Proc of the 39th IEEE/ACM International Syposium on Microarchitecture. 2006: 423-432.

[9] KIM S, CH D, SOLIHIN Y. Fair cache sharing and partitioning in a chip multiprocessor architecture [C]//Proc of the 13th International Conference on Parallel Architectures and Compilation Techniques. Washington DC: IEEE Computer Socitey, 2004: 111-122.

[10] DYBDAHL H, STENSTROM P. An adaptive shared/private NUCA cache partitioning scheme for chip multiprocessors [C]//Proc of the 13th International Symposium on High Performance Computer Architecture. 2007: 2-12.

[11] HERRERO E, GONZALEZ J, CANAL R. Elastic cooperative caching: an autonomous dynamically adaptive memory hierarchy for chip multiprocessors [C]//Proc of the 37th International Symposium on Computer Architecture. 2010: 419-428.

[12] HERRERO E, GONZALEZ J, CANAL R. Distributed cooperative caching [C]//Proc of the 17th International Conference on Parallel Architectures and Compilation Techniques. New York: ACM Press, 2008: 134-143.

[13] CHANG J, SOHI G S. Cooperative caching for chip multiprocessors [C]//Proc of the 33rd Internatioanl Symposium on Computer Architecture. 2006: 264-276.

[14] ZHANG M, ASANOVIC K. Victim replication: maximizing capacity while hiding wire delay in tiled chip multiprocessors [C]//Proc of the 32nd International Symposium Computer Architecture. 2005: 336-345.

[15] ZHANG M, ASANOVIC K. Victim migration: dynamically adapting between private and shared CMP caches, MIT-CSAIL-TR-2005-064 [R]. Cambridge: MIT Computer Science & Artifical Interlligence Laboratory, 2005.

[16] MARTIN M M K, SORIN D J, BECKMNN B M, *et al*. Multifacets general execution-driven multiprocessor simulator (GEMS) toolset [J]. SIGARCH Computer Architecture News, 2005, 33(4): 92-99.

[17] MAGNUSSON P S, CHRISTENSSON M, ESKILSON J, *et al*. Simics: a full system simulation platform[J]. Computer, 2002, 35(2): 50-58.

(上接第 219 页)

[12] 汤赐,姚舜,帅智康,等.新型注入式混合有源滤波器的稳定性研究[J].电网技术,2006,30(20):56-60.

[13] 曾令全,曾德俊,吴杰,等.用于有源滤波器谐波检测的一种新的自适应算法[J].电网技术,2008,32(13):40-44.

[14] 史伟伟,蒋全,胡敏强,等.低通滤波器在串联型电力有源滤波器中的应用[J].电网技术,2002,26(5):44-48.

[15] 戴朝波,林海雪.两种谐波电流检测方法的比较研究[J].中国电机工程学报,2002,22(1):81-85.

[16] 曾令全,白志亮,曾德俊,等.基于自适应神经网络的有源电力滤波器谐波电流提取方法[J].电力自动化设备,2010,10(2):45-40.

[17] 刘国海,吕汉闻,刘颖,等.基于改进 RLS 算法的谐波电流检测方法[J].电力自动化设备,2010,30(10):46-49.