Journal of Computer Applications
计算机应用,2013,33(9):2404 - 2409
ISSN 1001-9081
CODEN JYIIDU
2013-09-01
http://www.joca.cn

文章编号:1001-9081(2013)09-2404-06
doi:10.11772/j.issn.1001-9081.2013.09.2404

# 面向低功耗的多核处理器 Cache 设计方法

方 娟*, 郭 媚, 杜文娟, 雷 鼎
(北京工业大学 计算机学院, 北京 100124)
(*通信作者电子邮箱 fangjuan@bjut.edu.cn)

**摘 要**:针对多核处理器下的共享二级缓存(L2 Cache)提出了一种面向低功耗的 Cache 设计方案(LPD)。在 LPD 方案中,分别通过低功耗的共享 Cache 混合划分算法(LPHP)、可重构 Cache 算法(CRA)和基于 Cache 划分的路预测算法(WPP-L2)来达到降低 Cache 功耗的目的,同时保证系统的性能良好。在 LPHP 和 CRA 中,程序运行时动态地关闭 Cache 中空闲的 Cache 列,节省了对空闲列的访问功耗。在 WPP-L2 中,利用路预测技术在 Cache 访问前给出预测路信息,预测命中时则可用最短的访问延时和最少的访问功耗完成 Cache 访问;预测失效时,则结合 Cache 划分策略,降低由路预测失效导致的额外功耗开销。通过 SPEC2000 测试程序验证,与传统使用最近最少使用(LRU)替换策略的共享 L2 Cache 相比,本方案提出的三种算法虽然对程序执行时间稍有影响,但分别节省了 20.5%、17% 和 64.6% 的平均 L2 Cache 访问功耗,甚至还提高了系统吞吐率。实验表明,所提方法在保持系统性能的同时可以显著降低多核处理器的功耗。

**关键词**:片上多核处理器;二级缓存;动态划分;低功耗;性能
**中图分类号**:TP393 **文献标志码**:A

## Low-power oriented cache design for multi-core processor

FANG Juan*, GUO Mei, DU Wenjuan, LEI Ding
(*College of Computer Science, Beijing University of Technology, Beijing* 100124, *China*)

**Abstract:** This paper proposed a Low-Power oriented cache Design (LPD) of Level 2 (L2) cache for multi-core processors. LPD considered three different ways to reduce the power consumption while promising the best performance: Low Power oriented Hybrid cache Partition algorithm (LPHP), Cache Reconfiguration Algorithm (CRA), and Way-Prediction based on L2 cache Partition algorithm (WPP-L2). LPHP and CRA closed the columns that were not in use dynamically. WPP-L2 predicted one appropriate way before cache accesses, which could save the access time, so as to save power. These three methods of LPD saved power consumption by 20.5%, 17% and 64.6% on average over the traditional Least Recently Used (LRU) strategy with improvement of the throughput and little influence on the runtime of programs. The experimental results show that this method can reduce the power of multi-core processors significantly and maintain the system performance.

**Key words:** Chip Multi-core Processor (CMP); Level 2 (L2) cache; dynamic partition; low power; performance

## 0 引言

随着半导体技术水平的不断提高,片上集成的处理器核数也在不断地增加。单核处理器的性能已经近于极限,因此,多核乃至众核处理器已经取代传统的单核处理器成为处理器发展趋势的主流。多核处理器架构简单,设计易于优化和扩展。但是,目前与多核处理器相关的技术发展还未成熟,多核处理器潜在的优势尚未被完全挖掘出来。多核领域中还存在很多问题需要进一步的研究。

Cache 的大容量,快速及频繁的访问,使其访问功耗在片上处理器芯片的整体功耗中占主导地位,所占比例为 30% ~60%[1-2]。多核处理器系统功耗节省方案可在不同的系统层次上实现,以微处理器设计为例,在系统结构级上进行能源优化的比例为 40% ~90%,而在寄存器重定向级进行能源优化所占的比例为 15% ~30%。从系统结构级上提出的低功耗方案多集中在对 Cache 的设计。目前,在减少 Cache 功耗与提高共享 L2 Cache 性能方面已经提出了很多的算法方案[3-6]。

在保证共享 Cache 性能的基础上,本文提出了三种方法用于降低 Cache 的功耗:可重构 Cache、Cache 划分和 Cache 路预测。

## 1 LPD 相关技术

LPD 是指面向低功耗的 Cache 设计(Low Power Oriented-Cache Design)。多核处理器的共享 L2 Cache 通常拥有较大的容量,但其利用率却不高。因此,适当地减少 L2 Cache 的容量不仅可提高其利用率,还可以节省能耗的开销。Cache 的划分策略可以有效地解决对共享 Cache 竞争使用造成的访问冲突,这对高性能计算具有重要意义,其中以基于 Cache 路的划分所需系统开销最小。因此,本文提出的 Cache 划分算法就是以 Cache 列为单位进行划分。面向低功耗的混合划分算法(Low Power oriented Hybrid cache Partition algorithm,

**收稿日期**:2013-04-07;**修回日期**:2013-05-14。
**基金项目**:国家自然科学基金资助项目(61202076);北京市教委科技计划项目(KM201210005022)。
**作者简介**:方娟(1973 -),女,辽宁丹东人,副教授,CCF 会员,主要研究方向:计算机系统结构、多核计算; 郭媚(1988 -),女,江西赣州人,硕士研究生,主要研究方向:多核计算; 杜文娟(1986 -),女,河北石家庄人,硕士研究生,主要研究方向:多核计算; 雷鼎(1986 -),男,湖北荆州人,硕士研究生,主要研究方向:多核计算。

《计算机应用》Journal of Computer Applications

LPHP)主要利用程序执行时在时间和空间上的局部性原理,通过合并在 L2 Cache 访问中差异度较大的两个线程,并以此作为 Cache 划分的一个单位对共享 L2 Cache 进行动态划分。与以往的 Cache 划分策略相比,LHPH 算法可以节省更多的 Cache 路空间,即关闭更多的 Cache 路,从而更大程度地降低了 Cache 访问功耗。

Cache 可重构技术是指在程序运行过程中,动态地关闭程序运行时没有使用到的 Cache 资源,从而降低 Cache 的功耗。本文中提到的 Cache 可重构是指路可重构,通过动态监测程序执行过程中对 Cache 路的访问情况,关闭部分空闲的 Cache 路。Cache 可重构机制通过在 Cache 访问模块中增加可重构 Cache 算法,在硬件上增加可重构 Cache 的模块,使程序在运行过程中会改变 Cache 的相联度。

在基于 Cache 划分的路预测算法(WPP-L2)中,通过 Cache 划分策略将共享 L2 Cache 划分给每个处理器核,使得每个处理器核独占部分的 L2 Cache 空间。在进行 L2 Cache 访问之前,由路预测预先提供可能访问到的 Cache 路编号,L2 Cache 访问时直接读取预测路对应的标记和数据,如果路预测命中,则可在最短的访问延时内完成 L2 Cache 访问,同时节省了功耗。如果路预测失效,则在增加的时钟周期内只需读取该处理器核在划分时所得的 Cache 路中除预测路之外的剩余 Cache 路的标记和数据,减少了在预测失效时需要额外访问的 Cache 路的数量,从而降低了因预测失效导致的功耗开销。

## 2　LPD 策略

### 2.1　面向低功耗的混合划分策略 LPHP

通常情况下,Cache 划分策略可分为基于硬件的划分[8]和基于软件的划分[9]。基于硬件的划分有列划分[10],而基于软件的划分包括组划分[11]和页划分[12]。近期又有一些新的划分方案被提出,其划分的基本单位变得越来越小[13-14]。基于 Cache 列的划分策略逻辑简单清晰,并且能有效地避免线程间由于竞争 Cache 资源导致的冲突,因此 LPHP 将采用 Cache 列为基本单位进行共享 Cache 的划分。

#### 2.1.1　硬件结构

在四核处理器系统中,应用 LPHP 划分策略的共享 L2 Cache 的硬件结构如图 1 所示。其中缺失率监控器(Miss Rate Monitor, MRM)能够动态地收集每个处理器核中线程在不同 Cache 容量下运行时的缺失率, Cache 列中上半部分的访问次数(Cache Access to the Column's Upper half, ACU)和 Cache 列中下半部分的访问次数(Cache Access to the Column's Lower half, ACL),用来区分每个处理器核对 Cache 列的访问程度。

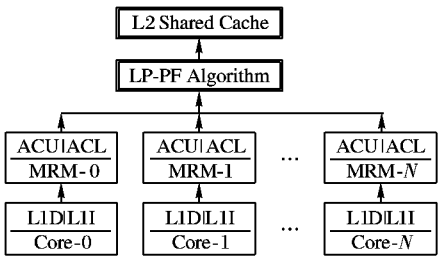


图 1　基于 LPHP 算法的共享 L2 Cache 结构

由于 LPHP 划分算法没有在内存访问的关键路径上,因此不会导致额外的时间开销。

#### 2.1.2　CMP 的性能模型

片上多核处理器(Chip Multi-Processor,CMP)的性能模型中以平均每周期执行的指令数(Instruction Per Cycle, IPC)为衡量标准,本文中对 IPC 的计算采用了 Matick 提出计算方法[15],如式(1)所示:

$$IPC = \frac{1}{CPI_{[base]} + E_1 + M_1 \times E_2 + M_2 \times E_3} \tag{1}$$

式(1)中假设对主存的访问不会发生缺失并且处理器最后一级的 Cache 为 L2 Cache。其中:$M_1$ 表示 L1 Cache 执行每条指令时的缺失数,$E_1$ 表示 L1 Cache 的命中开销,$M_2$ 表示 L2 Cache 执行每条指令时的缺失数,$E_2$ 表示当 L1 Cache 失效时访问 L2 Cache 的命中开销,$E_3$ 则表示当 L2 Cache 失效时访问主存的命中开销。

假设程序 $i$ 执行时 L2 Cache 的缺失率表示为 $\theta_i(x)$($x \in [0, C_{L2}]$),而划分给程序 $i$ 的 Cache 大小为 $x$,$C_{L2}$ 则表示 L2 Cache 的容量,则程序 $i$ 的性能指标 IPC 可由式(2)计算得到:

$$IPC_i = \frac{1}{CPI_{[base]} + E_1 + M_1 \times E_2 + \theta_i(c_i) \times E_3} \tag{2}$$

本文中的性能衡量即采用式(2)计算值获得。

#### 2.1.3　LPHP 划分的性能模型

根据划分所要达到目标的不同,Cache 划分策略可以分为三类:面向性能的划分策略[16]、面向公平性的划分策略[17]和面向服务(Quality of Service, QoS)的划分策略[18]。在 LPHP 算法中共有四个关键参数:差异度阈值 $R_{share}$、回溯阈值 $IPC_{[initial]}$、性能损失阈值 $PDT$ 和时间片 $T$。

定义公式 $f_i(k) = IPC_i(k)$,其中 $IPC_i(k)$ 表示程序 $i$ 在 $k$ 列 Cache 空间下执行时的 IPC。再进一步定义公式 $g_i(k) = f_i(k+1) - f_i(k)$,用来计算当程序 $i$ 划分所得的 Cache 空间由 $k$ 列增加到 $k+1$ 列时其 IPC 的增量。LPHP 算法的主要流程如下:

1)初始化阶段。

在本阶段中,以线程作为 Cache 划分单位,为每个线程划分一列 Cache 空间保证每个程序都有 Cache 资源支持其运行。在时间片 $T$ 结束后,计算系统性能指标 IPC 的值。如果 $IPC_{[par]} \geqslant (1 - PDT) \times IPC_{[initial]}$,则关闭剩余的 Cache 列,同时进入回溯阶段 3);否则进入合并阶段 2)。

2)合并阶段。

集合 $TS$ 包含所有未曾合并的线程,$TS = \{T_i \mid 1 \leqslant i \leqslant n\}$。

a)如果 $|TS| \geqslant 2$,进入步骤 b);否者进入步骤 e)。

b)通过式(3)计算 $TS$ 集合中每两个线程间的差异度 $R_{diff}$。

$$R_{diff} = |P_i - P_j|;\quad 1 \leqslant i \leqslant n; 1 \leqslant j \leqslant n;\ i \neq j \tag{3}$$

在 LPHP 算法中,每个 Cache 列被分为上半部分和下半部分两个部分。$U_{up}$ 表示一列 Cache 中上半部分被访问到的 Cache 块数,$U_{down}$ 则表示一列中下半部分被访问到的 Cache 块数。$P$ 表示线程的访问偏度,可由式(4)计算得出:

$$P = \frac{U_{up} - U_{down}}{U_{up} + U_{down}} \tag{4}$$

c)如果 $R_{diff} \geqslant R_{share}$,则将线程 $\langle T_i, T_j \rangle$ 合并为一个划分单元,同时将该线程对从集合 $TS$ 中除去,即 $TS = TS - \{T_i, T_j\}$。

d)如果 $|TS| \geqslant 2$,则进入步骤 c);否则进入步骤 e)。

e)如果仍然有多余的 Cache 列未被划分,则进入步骤 f);

Journal of Computer Applications 计算机应用

否则进入步骤 3)。

f)通过公式计算所有划分单位的 IPC 增量 $g_i(C_i)$。

g)划分一列 Cache 空间给 IPC 增量最大的划分单位。

h)通过式(2),计算系统性能 $IPC_{[par]}$。

i)如果 $IPC_{[par]} \geqslant (1 - PDT) \times IPC_{[initial]}$,则合并阶段结束。关闭剩余的 Cache 列同时进入步骤 3);否则进入步骤 e)。

3)回溯阶段。

a)当一个时间片 $T$ 结束后,计算性能指标 $IPC$ 的值 $IPC_{[par]}$。

b) 如果 $IPC_{[par]} \geqslant (1 - PDT) \times IPC_{[initial]}$,则进入步骤 3);否则回溯到初始阶段,即每个线程各分配一列 Cache 空间,然后进入步骤 2)。

最后关闭所有剩余的 Cache 列以节省 Cache 的访问功耗。

上述 LPHP 算法中虽然增加了 MRM,ACU 和 ACL 寄存器,在一定程度上对系统的功耗和运行时间都有所影响,但是相对于关闭的 Cache 列,系统的总体功耗还是有了大幅度下降。从总体运行时间来看,由于减小了 Cache 的搜索空间,程序的总体运行时间也有一定程度的下降。

### 2.2 Cache 可重构算法 CRA

#### 2.2.1 硬件结构

Cache 的可重构机制是基于软件模拟的,在 L2 Cache 模块中加入重构算法,因此可以在程序运行的过程中改变 Cache 的相联度。在程序初始运行时给 Cache 设置最优化的配置参数,且此时的参数已经是最大值,因此,在 Cache 的重构过程中其相联度不会再超过初始值。可重构 Cache 的硬件结构如图 2 所示。

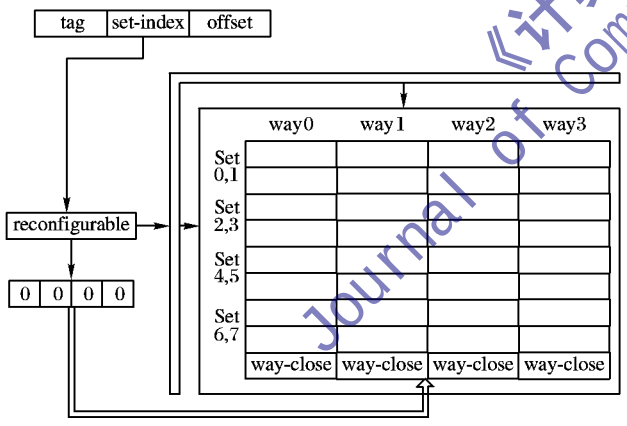


图 2 可重构 Cache 的结构

#### 2.2.2 算法结构

在重构算法中设定了一个时间周期,每当一个周期结束就判断是否需要进行 Cache 重构,如果需要重构 Cache,则调整 L2 Cache 的相联度。在本文提出的可重构算法中定义了一些参数和常量。其中,常量包括 *Reconfigurable_cycle*、$M_1$、$M_2$、*r_a_yu* 和 *r_a_low*。

*Reconfigurable_cycle* 定义了重构周期;$M_1$ 表示 L2 Cache 访问命中率的阈值;$M_2$ 表示 L2 Cache 访问次数的阈值,用于判断 Cache 访问次数是否过少;*r_a_yu* 表示 Cache 路最近访问次数的高阈值,表明当前的 Cache 路最近访问较为频繁;*r_a_low* 表示 Cache 路最近访问次数的低阈值,表明当前 Cache 路最近不经常被访问。重构算法中的相关参数包括 *last_cycle*、*hit_count*、*a*3、*a*2、*a*1 和 *a*0。其中:*last_cycle* 用于统计程序执行经过的重构周期数,其初始值为零;*hit_count* 表示当前 L2 Cache 的命中率;*a*0 ~ *a*3 分别表示当前周期内对应 Cache 路的访问次数,它们的初始值也是零。上述重构参数在重构周期开始时都会被重新清零。根据上述参数,L2 Cache 的命中率可以由式(5)计算:

$$hit\_rate = \frac{hit\_count}{a3 + a2 + a1 + a0} \tag{5}$$

而 L2 Cache 所有路的平均访问次数可以由式(6)计算:

$$way\_average\_count = \frac{a3 + a2 + a1 + a0}{4} \tag{6}$$

在重构 Cache 结构图中显示了控制状态的 4 个 0,它是用 4 位二进制的数字显示当前 Cache 路的开启情况,每一位对应着 L2 Cache 的每一路的情况。同时,用 key-value 的映射方式给每一个 Cache 路增加一个 value 值,为了统计各个路最近访问的情况,value 值分别是 *r_a*3、*r_a*2、*r_a*1、*r_a*0。

在重构周期内,比较当前 L2 Cache 的命中率与命中率阈值 $M_1$,如果当前的命中率高于 $M_1$,代表此时 Cache 的命中率很高,如果此时 Cache 访问次数低于访问次数的阈值,则表示当前对 Cache 空间的需求不高,可以关闭部分 Cache 路以降低 Cache 的功耗。如果当前 L2 Cache 的命中率过低,且没有达到预期的值,则需要考虑重新开放部分之前关闭的 Cache 路空间。

如果 L2 Cache 的命中率高于命中率的高阈值,则表示可以对 Cache 进行重构,适当地减小 Cache 空间;若 L2 Cache 命中率低于命中率的低阈值时,则表示需要适当地增大 Cache 空间。但是,在对 L2 Cache 的相联度进行改变时需要事先判断当前的相联度是否已经是最大或者最小了,如果相联度已经是最大值了,就无需进行重构了。

### 2.3 基于 Cache 划分的路预测算法 WPP-L2

#### 2.3.1 硬件结构

下面将介绍一种新的 L2 Cache 结构用于降低 L2 Cache 的访问功耗,即基于 Cache 划分的路预测 L2 Cache。基于 Cache 划分的路预测 Cache(WPP-L2 Cache)结构与传统的 L2 Cache 相比,增加了三个逻辑模块,分别为路预测器、路预测表和划分表;与传统的路预测 L2 Cache 相比增加了路划分表模块。WPP-L2 Cache 的结构如图 3 所示。

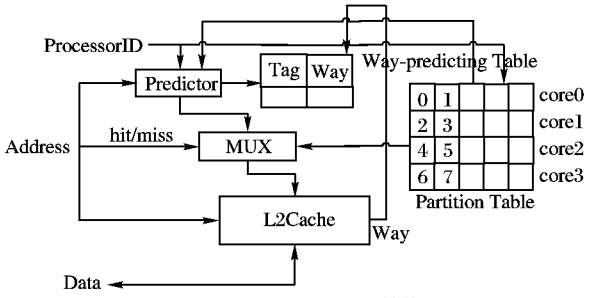


图 3 WPP-L2 Cache 的结构

#### 2.3.2 WPP-L2 算法

在 L2 Cache 访问前,将发起访问的处理器核编号与访问地址送往划分表和路预测表,用访问地址与路预测表内的 tag 项一一比较,如果找到匹配的地址,则读取对应的 Cache 路编号并将预测路信息送往 MUX。MUX 直接使能预测路,读取对应的 Cache 块的标记和数据,如果预测命中则可以在一个时钟周期内完成 L2 Cache 访问,并且只读取了一路 Cache 数据。如果路预测失效,则根据处理器核编号查找划分表,确定该处理器核通过划分所得到的 Cache 路编号,失效增加的时钟周期内读取属于该处理器核的除预测路之外的剩余 Cache

《计算机应用》Journal of Computer Applications

路的标记和数据,并从中找到需要访问的数据。

组相联 Cache 的访问功耗 $E_{cache}$ 可以由式(7)[19]来表示:

$$E_{cache} = (E_{Tag} \times N_{Tag} + E_{Data} \times N_{Data}) \quad (7)$$

其中:$N_{Tag}$ 和 $N_{Data}$ 分别表示 Cache 访问到的标记阵列和数据阵列的数量,$E_{Tag}$ 和 $E_{Data}$ 则分别表示访问一个标记和一个数据时消耗的能量。因此,WPP-L2 Cache 访问所消耗的能量($E_{WPPI2Cache}$)可以由式(8)计算:

$$E_{WPPI2Cache} = (E_{Tag} + E_{Data}) + (1 - PHR) \times \{(way\_count - 1) \times E_{Tag} + (way\_count - 1) \times E_{Data}\} \quad (8)$$

其中:*PHR* 表示路预测的命中率,*way_count* 则表示处理器核由 Cache 划分后所得到的 Cache 路数量。传统路预测 L2 Cache(WP-L2 Cache)的访问能耗则可由式(9)给出:

$$E_{WPI2Cache} = (E_{Tag} + E_{Data}) + (1 - PHR) \times \{(N_{Tag} - 1) \times E_{Tag} + (N_{Data} - 1) \times E_{Data}\} \quad (9)$$

由式(7)~(8)可以反映出采用 WPP-L2 算法 Cache 相比传统的 L2 Cache 其访问功耗减少了;Cache 划分不可能将所有的 Cache 路划分给一个处理器核,那么式(8)中的 *way_count* 必定小于式(9)中的 $N$,故路预测失效时,采用 WPP-L2 算法的 Cache 比采用传统路预测算法的 Cache 节省了能量开销。

## 3　模拟结果与分析

### 3.1　模拟环境

本文采用 Virtutech Simics[20]模拟器来模拟实现 4 核和 16 核处理器系统。Virtutech Simics 是一个开放、灵活的全系统模拟器,它能够模拟多种目标平台的系统架构。GEMS(General Execution-driven Multi-processor Simulator)是基于 Simics 的一组多核模拟器模块,它使得对目标系统的模拟更加具体。在模拟的多核处理器系统中,每个处理器核私有一个 L1 数据 Cache 和一个 L1 指令 Cache,所有核共享一个 L2 Cache。模拟的 4 核处理器系统配置和 16 核处理器的系统配置如表 1~2 所示。

**表 1　4 核处理器模拟配置**

| 参数 | 配置值 |
|---|---|
| Processor | 2.2 GHz, 0.5 ns cycle time, 4-core |
| L1 private data, Instruction Cache | 16 KB, 4-way, 64 B Cache line, LRU |
| L2 shared Cache | 1 MB, 16-way, 64 B Cache line, LRU |
| Main Memory | 2 GB, DDR2, 667 MHz |

**表 2　16 核处理器模拟配置**

| 参数 | 配置值 |
|---|---|
| Processor | 2.2 GHz, 0.5 ns cycle time, 16-core |
| L1 private data, Instruction Cache | 16 KB, 4-way, 64 B Cache line, LRU |
| L2 shared Cache | 2 MB, 32-way, 64 B Cache line, LRU |
| Main Memory | 2 GB, DDR2, 667 MHz |

因此,本文采用 CACTI6.5 功耗模拟器来评估本文提出的三个低功耗 Cache 算法中的 Cache 访问功耗。

### 3.2　实验结果

#### 3.2.1　LPHP 算法结果

为了验证 LPHP 算法的优势,本文分别模拟实现了在传统最近最少使用(Least Recently Used, LRU)替换策略下的 Cache 均分算法(Average Partition)和列划分算法(Column Partition),用于与 LPHP 算法进行比较。测试程序选取了 SPEC CPU2000 基准测试程序中的 gzip、bzip2、vortex、parser 和 vpr,其中 gzip、vpr 和 bzip2 是访问敏感的,而 vortex 和 parser 则是非访存敏感的[21]。三种划分算法下测试程序的运行时间如图 4 所示,此时选取的性能损失阈值 *PDT* 为 3%。

在传统的 LRU 替换策略中,每个处理器核划分得到 Cache 列的机会是均等的,基于 LRU 的划分策略属于静态划分。基于 Cache 列的划分算法可以节省功耗,但是它没有对线程进行合并。如图 4 所示,非访存敏感的程序在 LPHP 算法下的执行时间比其他两个划分算法要长,但是当处理器核数增加以后,这种差异并不明显。

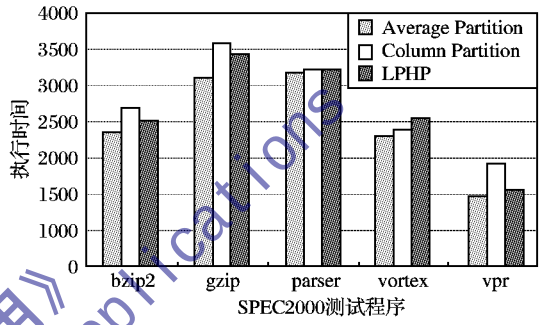


图 4　三种划分算法下测试程序的执行时间(16 核处理器)

图 5 显示了测试程序在 LPHP 算法和另外两种划分算法下执行时 L2 Cache 的功耗比较,LPHP 通过关闭剩余的 Cache 列来降低 L2 Cache 的功耗。从图 5 可看出,当 *PDT* 为 3%时,LPHP 算法相比传统的 LRU 算法和基于 Cache 列的划分算法,其对 L2 Cache 功耗的节省效果是非常明显的,而且,当处理器核数增加时,LPHP 也能节省更多的功耗。在 4 核处理器系统下,LPHP 算法节省的 L2 Cache 功耗要比传统的 LRU 替换策略多 18%,比基于 Cache 列的划分算法多 11%;而在 16 核处理器系统下,LPHP 算法节省的 L2 Cache 功耗要比传统的 LRU 替换策略多 23%,比基于 Cache 列的划分算法多 14%。

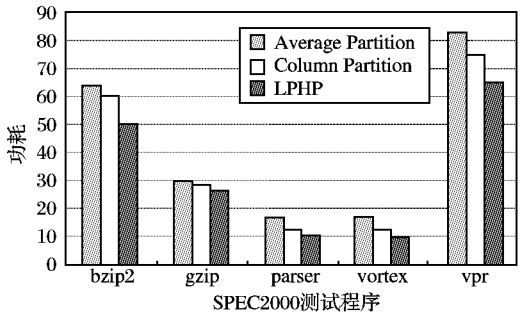


图 5　不同划分算法下 L2 Cache 功耗比较(16 核处理器)

#### 3.2.2　CRA 算法结果

图 6 显示了 SPEC CPU2000 测试程序在 16 核处理器系统下的可重构 Cache(r-Cache)和非可重构 Cache(s-Cache)下的运行时间。从图 6 可知,gzip 和 parser 两个测试程序在 r-Cache 下的执行时间要比在 s-Cache 下的执行时间要长,其原因是 Cache 可重构的过程增加了时间开销。

实验通过 Cache 相联度的改变和 Cache 的关闭率来计算 Cache 功耗的节省效率。如果 Cache 未进行重构则其相联度是不变的,重构时相联度会发生改变,本文通过计算 Cache 相

Journal of 《计算机应用》 Computer Applications

联度与程序执行时间之积来表示 Cache 的功耗开销。图 7 显示了 4 核和 16 核处理器系统下可重构 Cache 和非可重构 Cache 在执行测试程序时的功耗开销。从图 7 可知,16 核下的可重构 Cache 的功耗节省情况没有在 4 核处理器的明显,在 4 核处理器系统下,可重构 Cache 的功耗比非可重构 Cache 平均节省 17%。

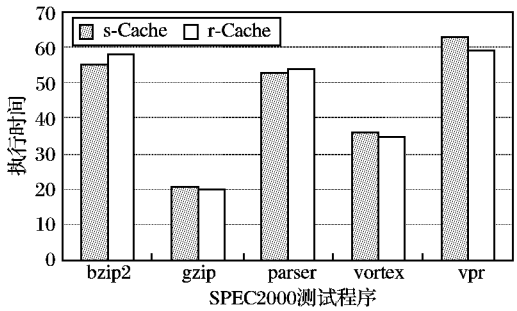


图 6 r-Cache 与 s-Cache 下测试程序执行时间(16 核处理器)

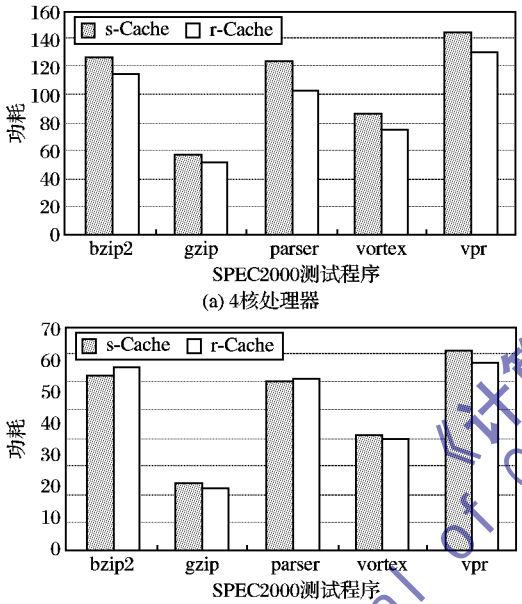


图 7 r-Cache 与 s-Cache 的功耗比较

### 3.2.3 WPP-L2 算法结果

为了验证 WPP-L2 算法的优势,本文选择了 5 组 SPEC CPU 2000 测试程序,分别在传统的 Conventional L2 Cache、基于路预测的 Way predicting L2 Cache 和 WPP-L2 Cache 下运行,其中 WPP-L2 Cache 中的划分算法选取了基于公平性的 DFC(Dynamic Fairness Caching)算法[22]。图 8 显示了在 4 核和 16 核处理器系统下测试程序在三种 L2 Cache 结构下的执行时间。图 9 显示了在 4 核和 16 核处理器系统下测试程序在三种 L2 Cache 结构下的执行时的功耗开销。

从图 8~9 中显示,WPP-L2 算法下的测试程序执行时间比传统的路预测 L2 Cache 算法平均增加 6.9%,比传统的 L2 Cache 下的程序执行时间平均增加 27.5%;但是 WPP-L2 算法下执行测试程序时节省的 Cache 访问功耗要比传统的路预测 L2 Cache 算法下的访问功耗平均降低 25.6%,比传统的 L2 Cache 下的程序执行时访问功耗平均降低 64.6%。通过 CACTI6.5 模拟得出每次访问 WPP-L2 Cache 时由查找路预测表和路划分表而导致的额外功耗开销比传统的 L2 Cache 平均增加 0.2%。而由 WPP-L2 节省的 Cache 功耗要远大于路预测表和路划分表所增加的额外功耗。

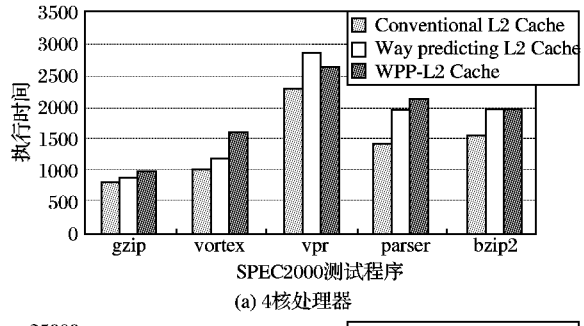


图 8 测试程序在三种 L2 Cache 算法下的执行时间

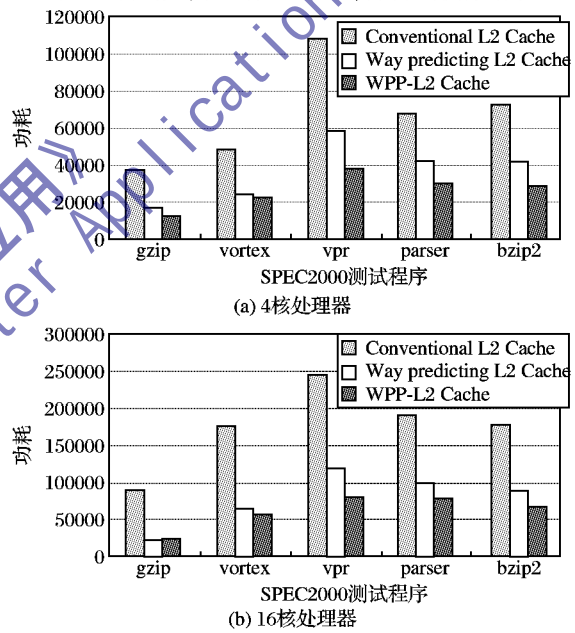


图 9 三种 L2 Cache 算法下测试程序执行的功耗

## 4 结语

本文通过三种不同的方法实现了降低多核处理器下 L2 Cache 访问功耗的目的,分别是 Cache 混合划分算法,Cache 可重构算法和基于公平性 Cache 划分的路预测算法、LPHP 算法基于软件的划分且易于实现,在 LPHP 算法中,将 Cache 划分作为一个非线性动态规划问题,并设计了最佳的算法解决这一问题。在程序运行时通过将部分线程合并为一个划分单位,使得有很多的 Cache 列剩余,当关闭这些剩余 Cache 列时,Cache 的访问功耗就得到了降低;同时通过设置性能损失阈值,使得算法对 Cache 性能的影响很小。CRA 算法相对更加灵活,在程序运行过程中根据程序的运行时间等信息动态地改变 Cache 相联度,因为算法的复杂度更高,因此对 Cache 性能的影响相对较大。而 WPP-L2 算法则通过结合 Cache 划分算法与路预测算法的方式,来达到降低 Cache 功耗目的,它可以与 LPHP,CRA 或者其他 Cache 设计相结合使用。

总体来说,上述三种算法都不同程度地降低了 Cache 的访问功耗,表现出了良好的能量效率,对性能的影响也控制在

Journal of Computer Applications 《计算机应用》

可以接受的范围内。本文下一步的工作将主要集中在建立功耗与性能的关系模型,设计出更优的低功耗算法方案。

**参考文献:**


[1] GONZALEZ R, HOROWITZ M. Energy dissipation in general purpose microprocessors[J]. IEEE Journal of Solid State Circuits, 1996, 31(9):1277 – 1284.

[2] VIJAYKRISHMAN N, KANDEMIR M, IRWIN M J, *et al*. Energy-driven integrated hardware-software optimizations using simple-power[C]// ISCA-27: Proceedings of the 27th Annual International Symposium on Computer Architecture. New York: ACM Press, 2000:95 – 10.

[3] CALDER B, GRUNWALD D, EMER J. Predictive sequential associative cache[C]// Proceedings of the 2nd International Symposium on High-Performance Computer Architecture. New York: ACM Press, 1996: 244 – 253.

[4] CHEN H C, CHIANG J S. Low-power way-predicting cache using valid-bit predecision for parallel architectures[C]// Proceedings of the 19th International Conference on Advanced Information Networking and Applications. Washington, DC: IEEE Computer Society, 2005: 203 – 206.

[5] ZHANG C, ZHANG X, YAN Y. Two fast and high-associativity cache schemes[J]. IEEE Micro, 1997, 17(5):40 – 49.

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

[7] PWELL M D, AGARWAL A, VIJAYKUMAR T N, *et al*. Reducing set-associative cache energy via way-prediction and selective direct-mapping[C]// Proceedings of the 34th Proceedings ACM/IEEE International Symposium on Micro-Architecture. Washington, DC: IEEE Computer Society, 2001: 54 – 65.

[8] CHAUDHURI M. Page NUCA: selected policies for page grain locality management in large shared chip multiprocessor caches[C]// Proceedings of HPCA-15. Washington, DC: IEEE Computer Society, 2009:227 – 238.

[9] AWASTHI M, SUDAN K, BALASUBRAMONIAN R, *et al*. Dynamic hardware — assisted software controlled page placement to manage capacity allocation and sharing within large caches[C]// Proceedings of HPCA-15. Washington, DC: IEEE Computer Society, 2009: 250 – 261.

[10] CHANG J. Cooperative caching for chip multi-processors[D]. Madison, Wisconsin, USA: University of Wisconsin at Madison, 2007.

[11] LIU C, SIYASUBRAMANIAM A, KANDEMIR M. Organizing the last line of defense before hitting the memory wall for CMPs[C]// Proceedings of the 10th International Symposium on High Performance Computer Architecture. Washington, DC: IEEE Computer Society, 2004: 176 – 185.

[12] LIN J, LU Q, DING X, *et al*. Gaining insights into multi-core cache partitioning: bridging the gap between simulation and real systems[C]// HPCA 2008: Proceedings of the 14th IEEE International Symposium on High Performance Computer Architecture. Piscataway, NJ: IEEE Press, 2008: 367 – 378.

[13] SANCHEZ D, KOZYRAKIS C. Vantage: scalable and efficient fine-grain cache partitioning[J]. Computer Architecture News, 2011, 39(3): 57 – 68.

[14] HASENPLAUGH W, AHUJA P S, JALEEL A, *et al*. The gradient-based cache partitioning algorithm[J]. ACM Transactions on Architecture and Code Optimization, 2012, 8(4):1 – 20.

[15] 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 – 843.

[16] 所光, 杨学军. 面向多线程多道程序加权共享 Cache 划分[J]. 计算机学报, 2008,33(11): 1938 – 1946.

[17] QURESHI M K, PATT Y N. Utility based cache partitioning: a low overhead, high performance, runtime mechanism to partition shared caches[C]// MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. Washington, DC: IEEE Computer Society, 2006:423 – 432.

[18] 隋秀峰, 吴俊敏, 陈国良. ARP: 同时多线程处理器中共享 Cache 自适应运行时划分机制[J]. 计算机研究与发展, 2008, 45(7):1269 – 1277.

[19] IYER R. CQoS: a framework for enabling QoS in shared caches of CMP platforms[C]// ICS 2004: Proceedings of the 18th Annual International Conference on Supercomputing. New York: ACM Press, 2004:257 – 266.

[20] PAUL M, PETROV P. Dynamically adaptive I-cache partitioning for energy-efficient embedded multitasking[J]. IEEE Transactions on Very Large Scale Integration Systems, 2011, 19(11):2067 – 2080.

[21] INOUE K, ISHIHARA T, MURAKAMI K. Way-predicting set-associative cache for high performance and low energy consumption[C]// ISLPED 1999: Proceedings of the 1999 International Symposium on Low Power Electronics and Design. New York: ACM Press, 1999: 273 – 275.

[22] 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 2403 页)


[4] WANG S, YANG Y. Fault tolerance in bubble-sort graph networks[J]. Theoretical Computer Science, 2012, 421: 62 – 69.

[5] ZHANG Z, XIONG W, YANG W. A kind of conditional fault tolerance of alternating group graphs[J]. Information Processing Letters, 2010, 110(22): 998 – 1002.

[6] YANG Y, WANG S. Conditional connectivity of star graph networks under embedding restriction[J]. Information Sciences, 2012, 199: 187 – 192.

[7] BOSE B, BROEG B, KWON Y, *et al*. Lee distance and topological propertices of *k*-ary *n*-cubes[J]. IEEE Transactions on Computers, 1995, 44(8): 1021 – 1030.

[8] GHOZATI S A, WASSERMAN H C. The *k*-ary *n*-cube network: modeling, topological properties and routing strategies[J]. Computers and Electrical Engineering, 1999, 25(3): 155 – 168.

[9] STEWART I A, XIANG Y. Bipanconnectivity and bipancyclicity in *k*-ary *n*-cubes[J]. IEEE Transactions on Parallel and Distributed Systems, 2009, 20(1): 25 – 33.

[10] ADIGA N R, BLUMRICH M A, CHEN D, *et al*. Blue Gene/L torus interconnection network[J]. IBM Journal of Research and Development, 2005, 49(2): 265 – 276.

[11] WANG S, ZHANG G, FENG K. Fault tolerance in *k*-ary *n*-cube networks[J]. Theoretical Computer Science, 2012, 460: 34 – 41.

[12] BONDY J A, MURTY U S R. Graph theory[M]. Berlin: Springer, 2008.


Journal of Computer Applications 计算机应用