

多核处理器片上可重构 Cache 系统及其机制设计

谢 憬, 章 裕, 王 琴, 毛志刚

(上海交通大学 电子信息与电气工程学院, 上海 200240)

摘 要: 针对多核处理器规模化数据访存与并行线程交叉数据使用的特性, 提出了一种可重构 Cache 的设计方案, 包含其基本硬件逻辑结构和工作机制; 同时提出了一种可在线动态重构 Cache 结构配置字生成的 DCAC 配置方法。实验证明, 上述设计方案配合在线配置方法工作, 能有效实现多核处理器系统根据不同的应用实时地配置共享 Cache 的组相联度, 使得近处理器内核的 Cache 系统有效提升了命中率, 在硬件开销增加 4.07% 的情况下, 缺失代价平均下降约 16.13%, 从而达到了多核处理器性能优化的目标。

关键词: 多核; 可重构; Cache; 配置; 缺失代价

中图分类号: TN4

文献标识码: A

文章编号: 1000—7180(2016)12—0001—05

The Design of Reconfigurable Cache Scheme in Multi-core Processor

XIE Jing, ZHANG Yu, WANG Qin, MAO Zhi-gang

(Department of Microelectronics and Nanoscience, Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract: To deal with massive and parallel data processing, the paper proposed a design solution of reconfigurable cache for multi-core processor. The work included the design of reconfigurable cache structure and its operation scheme, as well as a reconfiguration context word generation method, named as Dynamic Cache Associativity Configuration (DCAC) method. The experiments verified that with 4.07% overhead in hardware cost, the design solution won a 16.13% reduction of miss penalty in average for the core nearby caches with dynamic cache associativity reconfiguration method.

Key words: multi-core; reconfigurable; cache; configuration; miss penalty

1 引言

随着集成电路工艺的不断提高, 出现了高性能多核系统结构。多核处理器往往具备拥有多核运算引擎的特点。典型的多核处理器中, 多个运算核通过统一调度, 并且通过统一的寻址方式来共享主存, 即其具有线程级并行任务执行的特征。而随着片上处理器内核的个数不断增加, 使得存储器的带宽成为影响系统性能的关键瓶颈之一^[1]。

典型片上多核系统会共享某一层或者多层 Cache, 而访存密集型的应用则将大幅增加对 Cache 的访问频率。在运行多线程程序的过程中, 多核处理

器需要支持运行交叉执行各个线程程序^[2]。程序执行过程中, 应用表现出的阶段性特征, 更期望相应的软硬件能实现对 Cache 资源的动态调整配置, 从而尽可能高地发挥出系统的性能。

可重构 Cache 技术能够通过结构、映射方式的在线重构实现 Cache 访存机制与性能的优化。一个组相联方式高效和灵活的可重构 Cache 设计方案, 能够提高 Cache 的命中率, 并且减少 Cache 的缺失代价。而这也是当前新型高性能多核处理器设计技术研究的重要方向之一^[3]。

本文的主要工作, 是通过增加可重构 Cache 的 Tag 位比较电路, 并利用运算符模块划分思路, 配合

动态 Cache 重构配置字生成方法,实现 Cache 组相联方式面向特定应用的最佳在线重构,进而达到大规模应用中获取性能优化的目标。

2 高性能多核处理器可重构 Cache 设计方案

2.1 Tag 位比较电路机制改进

典型共享 Cache 的相联方式,其实现依赖于基于 Tag 位比较电路、逻辑电路、存储器和状态机等组成的复杂结构.如 Cache 总容量为 8 kB 时,每个 Cache Line 为 16 B,采用 4 路组相联的结构,则 Tag 电路的存储体共有 128 个标记位入口;标记位 21 位宽;每个 Tag 行有 1 个有效位,该位为 0 表明缺失,1 则表示命中. Tag 电路有预充电、加载和输出三种状态;写(Write)和比较(Compare)这两种控制信号用于控制 Tag 电路在上述状态之间转换。

在 CPU 的访问地址中,索引信号结合块内偏移经过译码,选通 Tag 阵列中的某一行,将被选中的标记位与 CPU 给出的 Tag 信息进行比较.若比较结果一致,即表明命中,访问数据经过灵敏放大器输出.基于一个 4 路组相联的 Cache 电路,则同时会有 4 个标记位与来自 CPU 的 Tag 进行对比,且 4 路中最多只有 1 路是命中的;如果缺失,则电路切换到加载状态,从下一级 Cache 中去取数.要实现可重构 Cache 的设计,就需要对 Cache 结构中的 Tag 位电路以及其控制电路部分做出相应调整,主要方案如下:

- 实现 Tag 位宽度可调整.例如,在直接映射,2 路组相联,4 路组相联和 8 路组相联的切换过程中,Tag 位宽分别为 18、19、20 和 21;为了实现可重构,需要对 Tag 的位宽做出实时调整。

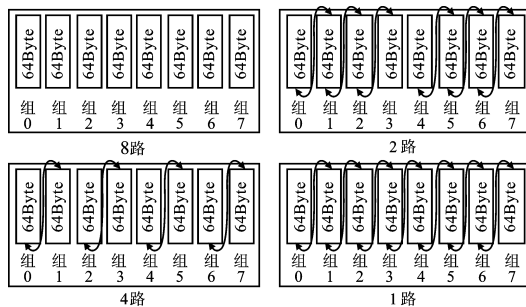
- 对 Tag 阵列的输出比较做出调整.在直接映射,2 路组相联,4 路组相联和 8 路组相联的切换过程中,Tag 位比较电路的输入数量分别为 1、2、4 和 8.而 Tag 比较电路的数量分别为 256、128、64 和 32。

2.2 硬件结构的调整

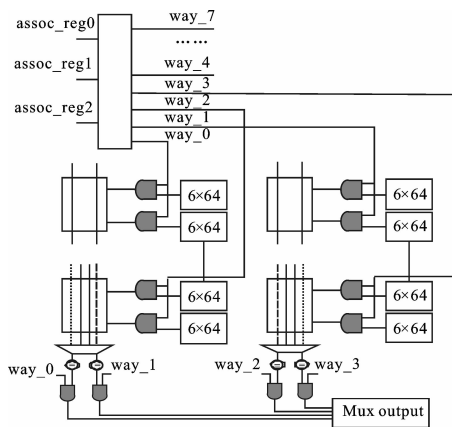
为了保证 Cache 总的使用容量不变,只改变组相联,其逻辑组织关系如图 1(a)所示形式,其中通过箭头连接的表示一个组,每个组内有若干个 Cache Line,而 Cache Line 的数量是由组相联度决定的^[4];高相联度 Cache 中每组的 Cache Line 数量多于低相联度每组的 Cache Line 的数量,而此时需要增加相关的控制逻辑来实现这一功能。

如图 1(b)所示,assoc_reg0, assoc_reg1, assoc_

reg2 是三个 1 位的寄存器,主要用于将 Cache 的相连方式配置成 8 路、4 路、2 路和 1 路组相联的结构.这两个寄存器经过译码生成 way_0, way_1, way_2, way_3 信号.当 assoc_reg0, assoc_reg1 和 assoc_reg2 都为 1 时,Cache 被配置成 8 路组相联,同时从 way_0 至 way_7 都被配置成 1,则 8 路的所有 Tag 比较输入都被激活;而 assoc_reg0, assoc_reg1, assoc_reg2 都为 0 时,Cache 被配置成 1 路组相联,那么 way_0 至 way_7 有且只有一路信号被配置成 1,而其余都为 0,因此 Tag 比较的输入中,只有一组被激活,同样 Tag 比较器也只有一个被激活使能;当 assoc_reg0=1, assoc_reg1 和 assoc_reg2 为 0 时,Cache 被配置成 2 路组相联,way_0 至 way_7 中有两个信号被置 1,在这种情况下,相应的电路结构通过解码器将两路的 Tag 阵列、数据阵列和 Tag 比较器激活使能.根据上述定义,通过在线配置,即可实现对 Cache 组相联结构的动态重构。



(a) 组相联重构形式



(b) 配置单元逻辑(以前四路为例)

图 1 可重构 Cache 设计要点

2.3 可重构辅助寄存器及在多核片上系统中的使用

在 Cache 的在线可重构实现过程中,CPU 需要捕获来自 Cache 的反馈信息,例如缺失次数、缺失率、缺失代价等,并以此作为控制重构配置字定义及

其生成的重要参照。本次设计中新增可重构(RC)辅助寄存器及其相应逻辑作为向主控 CPU 提供上述在线测算依据的硬件资源,该寄存器主要涵盖功能包括:

(1)指令计数器。该计数器用于当前周期中未执行的指令数量,每执行完一条指令,该计数器减一;由于相应的动态可重构配置程序需要在一定周期内完成一次重构,所以系统需要记录该周期内未执行的指令数量,提前为 Cache 准备好用于可重构的配置字;基于初始化配置信息,指令计数器能够协助计算重构配置字的时间;当计数器清零时,Cache 进行一次可重构^[5],其中重构请求是通过中断的形式发出的。

(2)访存计数器。访存计数器用于记录上一层 Cache 对本级 Cache 的访问次数,即上一层 Cache 的缺失次数。

(3)Cache 缺失计数器。该计数器用于记录可重构 Cache 的缺失次数,即程序执行过程中,其向下级存储索取数据的次数。在动态可重构配置程序中,主控 CPU 需要根据来自 Cache 的反馈来实时生成配置字信息,其中 Cache 的反馈信息则是通过 Cache 缺失计数器来生成的。

涵盖上述三个计数器功能的 RC 辅助寄存器,对于主控 CPU 是作为中断标志来使用的,即当指令计数器即将为零时,该寄存器由可重构 Cache 模块向 CPU 发出中断信号,从而使得 CPU 通过中断程序完成对 Cache 结构的重构、配置与调整。

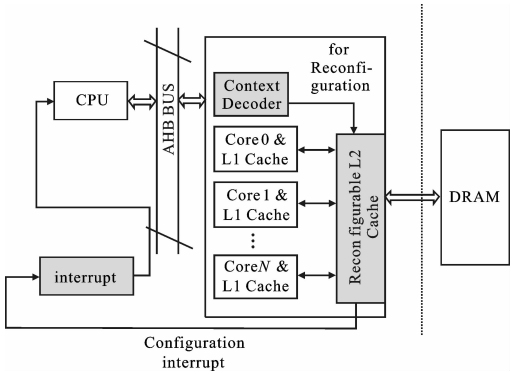


图 2 基于可重构 Cache 片上多核系统结构

如图 2 所示,片上多核系统可重构 L2 Cache 利用其 RC 辅助寄存器信息,通过在配置过程中向主控 CPU 发出中断信号,从而获得一次 CPU 的配置机会。而在中断子程序中,CPU 将根据 L2 Cache 中的反馈信息来生成合理的配置字,并在中断子程序执行过程中完成对 Cache 的配置和重构。

3 可重构 Cache 配置字生成方法

本文提出一种在线动态的 Cache 结构配置字生成 (Dynamic Cache Associativity Configuration, DCAC) 方法。DCAC 配置是一个基于缺失率和缺失代价为上下限、在线自适应调整的过程。缺失代价(Miss Penalty)、命中时间(Hit Time)、缺失率(Miss Rate)之间的关系如式(1)所示,其中的 L1、L2 和 L3 分别表示了三个 Cache 或者存储单元的层次。

$$\text{MissPenalty}_{L1toL2} = \text{HitTime}_{L1toL2} + \text{MissRate}_{L2} \times \text{MissPenalty}_{L2toL3} \quad (1)$$

如图 3 所示,DCAC 在配置初期,首先初始化 Cache 结构,并提供相应的初始化结构配置信息;每当完成一个应用运算周期,即各计数器达到某个特定值时(计数器清空之前),RC 辅助寄存器向 CPU 发出中断;进入重构配置中断子程序后,CPU 通过总线获取 Cache 的访问信息,其中通过获取 RC 辅助寄存器中的缺失数量计数器和访问次数计数器,用于计算得到缺失率信息。

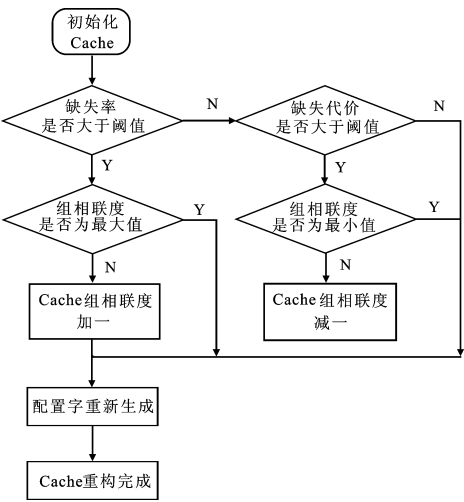


图 3 基于 DCAC 方法的可重构 Cache 配置字生成流程图

程序执行前,将在离线仿真(profiling)应用评估时表现最好的缺失率和缺失代价作为 DCAC 配置法中的阈值信息。当 Cache 的访问记录发现缺失率高于阈值的时候,此时主控 CPU 去检查 Cache 此时是否达到组相联度的最大配置值,若已达到,则确定此时的 Cache 结构为最佳,并将当前结构及其配置保持至下一个运算周期的运算;若未达到,则需增加 Cache 的组相联度,增加的幅度根据缺失率值来确定。其中当缺失率稍高于阈值,则组相联度增加一倍,例如从直接映射改变为 2 路组相联;当缺失率远高于阈值,则组相联度增加两倍,例如

从直接映射改变四路组相联. 当缺失代价大于阈值时,此时需查看 Cache 结构是否为最小组相联结构;当 Cache 未达到最低组相联度时,组相联度增加;当 Cache 结构已经达到最低组相联度时,确定此时为最佳结构,且保持当前结构及其配置信息至下一运算周期.

4 实验与分析

4.1 实验环境设置

文章选用 GEM5 为仿真平台,采用 Ruby 组件对可重构 Cache 进行了硬件建模,为配以 SPEC2000 作为主要实验 Benchmark. 多核架构的基本信息包括:1 个 32 bit ARMv7 指令内核作为主控 CPU 内核;4 个 ARMv7 指令内核作为应用运算内核模型;每个应用运算内核配备 1 个 32 kB 的 L1 指令 Cache 和 1 个 32 kB 的 L1 数据 Cache;系统共享一个 2 MB 容量 L2 Cache;同时对系统采用 LRU 替换策略和轮询仲裁策略. 另外采用 Verilog RTL 对可重构 Cache 系统进行了硬件逻辑设计.

由于本文可重构 Cache 设计方案中可改变的参数仅限于共享 Cache 的组相联度配置,因此实验主要关注对片上多核存储层次结构中 L2 Cache 的优化,即在实验过程中,将通过组相联度配置信息的动态变化来实现 Cache 的可重构,并评估其对系统整体的性能表现,同时测算其带来的硬件资源开销变化.

4.2 DCAC 配置字的生成与分析

如前所述,实验首先获得 profiling 下表现最好的缺失率和缺失代价作为 DCAC 配置法中的阈值信息,并配置于主控 CPU 通用寄存器中用于配置程序执行. 实验中以主控 CPU 的时钟周期为基本运算周期. 实验程序开始后,主控 CPU 基于 DCAC 方法在线动态生成可重构 Cache 配置字,并通过总线送到运算核的 Cache 配置字译码器端用于完成在线重构配置.

Cache 重构目的在于性能优化,但对对应代价是增加了额外的配置时间. 配置时间会影响到 Cache 的命中时间. 固定单一的 Cache 结构不存在配置时间消耗,而可重构 Cache 需要将这部分时间消耗纳入影响命中时间的范畴. 因此首先在 profiling 期间,根据应用性能最优化目标,在指定时间范围或者其他设置条件下,对目标应用过程中发生的 Cache 重构所带来的配置时间开销进行测算. 相应实验的应用程序选取了 SPEC2000 中的 Swim、Gzip、Gcc、Bzip2,并首先按照单核应用的性能最大化要求,获

取了在指定时间内通过重构使用不同组相联度 Cache 的次数,及其各自对应的单次配置时间消耗,其中也包含了 Tag 比较电路功能所花费的时间,具体如表 1 所示. 以 Swim 为例,在 DCAC 配置过程中,配置成直接映射需要花费 8 个时钟周期,而直接映射占用了 3 912 次,每次持续 1 000 000 000 个周期,总共花费 31 296 个时钟周期用于直接映射的配置;而配置成 2 路组相联结构需要花费 10 个周期,发生 8 107 次,即花费 81 070 个时钟周期用于 2 路组相联的配置.

表 1 各基础配置的占用次数

应用	变化方式	单次配置 耗时间/cycle	发生次数	总配置时间 /cycle
Swim	1way	8	3 912	31 296
	2way	10	8 107	81 070
	4way	9	43 416	390 744
	8way	8	760 920	6 087 360
	Total	—	816 355	6 590 470
Gzip	1way	8	562	4 496
	2way	10	771	7 710
	4way	9	320	2 880
	8way	8	4 519	36 152
	Total	—	6 172	51 238
Gcc	1way	8	8 052	64 416
	2way	10	3 196	31 960
	4way	9	4 339	39 051
	8way	8	78 019	624 152
	Total	—	93 606	759 579
Bzip2	1way	8	3 329	26 632
	2way	10	3 419	34 190
	4way	9	6 195	55 755
	8way	8	810 563	6 484 504
Total		—	823 506	6 601 081

4.3 性能分析与比较

从表 1 可以看出,评测的应用程序在运行中普遍对 8 路组相联 Cache 结构使用更为频繁,因此性能分析实验主要是将本文的基于可重构共享 L2 Cache 设计方案同基于 8 路固定组相联共享 L2 Cache 的方案进行比较,比较的主要指标为式(1)所定义的 L1 Cache 的缺失代价.

实验首先采用 4 个内核执行相同应用程序和不同应用程序两种模式执行. 从表 2 可以看出,在多核执行相同应用程序时,基于对应应用 DCAC 配置阈值设置,可重构 Cache 方案使得 L1 Cache 的缺失代价平均降低了 16. 13%.

其次实验令 4 个内核各自执行不同应用程序,分别使用表 2 中程序应用所得到的最高和最低 DCAC 配置阈值,并且以该应用对应内核的 L1 Cache 向共享 L2 Cache 的访存情况作为 DCAC 在线动态配置字生成过程中的所获取检验条件的信息来源,分别检测两种阈值设置下的系统性能情况.由于这其中的共享 L2 Cache 轮询时间消耗并不规则,因此其在性能表现上会受到影响.尽管如此,但从表 3 可以看出,基于可重构 L2 Cache 的表现仍优于固定 8 路组相联模式.同时表 3 也反映出在进行不同应用处理时,阈值越高,整体发生重构的次数越少,整个过程的缺失率偏高;阈值越低,整体发生重构的次数越多,命中率偏低.最终反应在缺失代价上,低阈值的表现更好,但也会因重构频繁带来其他诸如功耗等问题,对此如何平衡将是未来工作的研究重点.

表 2 各内核执行相同应用程序下的
L1 Cache 缺失代价对比

应用	DCAC 配置时 缺失代价 阈值设定	L2Cache 固定 8 路组 相联下 L1Cache 缺失代价 /cycle	基于可重构 L2 Cache 下 L1Cache 缺失代价 /cycle
Swim	15. 28	15. 28	10. 65
Gzip	2. 984	2. 984	3. 115
Gcc	4. 616	4. 616	4. 135
Bzip2	5. 784	5. 784	4. 865

表 3 各内核执行不同应用程序下的
L1 Cache 缺失代价对比

应用	L2Cache 固定 8 路组 相联的 L1 Cache 缺失 代价/cycle	基于可重构 L2 Cache 下 L1 Cache 缺失代价/cycle	
		DCAC 配置 使用最高应 用阈值设定 /15. 28	DCAC 配置 使用最低应 用阈值设定 /2. 984
Swim (Core0)	19. 02	12. 81	12. 01
Gzip (Core1)	5. 87	3. 415	3. 275
Gcc (Core2)	6. 76	4. 46	4. 266
Bzip2 (Core3)	6. 038	5. 435	4. 96

此外,通过 RTL 硬件设计,并结合 SMIC 65 nm 工艺及 Synopsys Design Compiler 工具进行综合,根据 2 MB 容量配置要求,实验的可重构 L2 Cache 系统及其相应逻辑电路的整体面积为 10 216 097 μm^2 ,较采用了基于固定 8 路组相联的 L2 Cache 结构电路(面积 9 816 202 μm^2)面积增加了约 4. 07%,硬件开销的增加基本可以接受.

5 结束语

本文针对多核处理器规模化数据访存与并行线

程交叉数据使用的特性,提出了可重构 Cache 的设计方案,并介绍了其基本硬件结构和工作机制;同时提出了用于在线动态生成 Cache 结构的配置字的 DCAC 配置方法.通过基于 GEM5 仿真环境和 SPEC2000 Benchmark 应用验证,该设计方案能有效实现多核处理器系统根据不同的应用实时地配置共享 Cache 的组相联度,使得近处理器内核的 Cache 系统有效提升了命中率,在系统硬件开销增加 4. 07%的情况下,缺失代价平均下降约 16. 13%,从而达到了多核处理器性能优化的目标.

参考文献:

[1] Modarressi M, Hessabi S, Goudarzi M. A reconfigurable Cache architecture for object-oriented embedded systems[C]//Electrical and Computer Engineering, CCECE'06. Canadian, IEEE, 2006: 959-962.

[2] Manakkadu S, Dutta S. Bandwidth based performance optimization of Multi-threaded applications[C]//Parallel Architectures, Algorithms and Programming (PAAP), 2014 Sixth International Symposium on. China, Beijing, IEEE, 2014: 118-122.

[3] Avakian A, Nafziger J, Panda A, et al. A reconfigurable architecture for multicore systems[C]//Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on. USA:Atlanta, IEEE, 2010:1-8.

[4] Zhang C, Vahid F, Najjar W. A highly configurable Cache architecture for embedded systems[C]//Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on. USA, San Diegeo, IEEE, 2003: 136-146.

[5] Gordon-Ross A, Vahid F, Dutt N D. Fast configurable-Cache tuning with a unified second-level Cache [J]. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2009, 17(1): 80-91.

作者简介:

谢 憬 男,(1981-),博士研究生,助理研究员.研究方向为大规模成电路设计、片上处理器体系结构、先进集成电路设计方法. E-mail: xie_xiejing@126. com.

章 裕 男,(1990-),硕士研究生.研究方向为大规模成电路设计、片上处理器体系结构.

王 琴 女,(1975-),博士,副教授.研究方向为大规模集成电路设计、片上处理器体系结构、先进集成电路设计方法.

毛志刚 男,(1962-),博士,教授,博士生导师.研究方向为大规模集成电路设计、可重构体系结构设计、集成电路与元器件可靠性等.