

共享指令缓存 XOR 散列索引的研究与设计

刘 晓 唐 勇 郑 方 丁亚军

(江南计算技术研究所 江苏 无锡 214083)

摘 要 SPMD(Single Program Multiple Data)是高性能领域的主要工作模式之一,该模式下邻近核心执行相同的程序块,但根据处理数据或控制流的差异,临近核心的指令流并不完全相同. L1 ICache(Instruction Cache)共享技术通过将邻近核心的 L1 ICache 共享,能有效利用众核处理器 SPMD 工作模式的特点,同时能缓解片上资源紧张的问题,但共享结构会带来访问冲突,对性能有不利影响. 本文基于排队网络对共享 ICache 的访问冲突进行了理论分析,该理论分析依据核心对共享 ICache 体的访问特性进行建模,避免了直接抽象物理节点导致的模型访存特性模糊问题. 根据理论推导的指令缓存性能损失原因,本文设计了面向共享 L1 ICache 的低访问冲突 XOR 散列函数. 函数的设计综合考虑搜索了代价和工程实现复杂性,在保证散列线性空间随机散列能力的前提下,对附加延迟、功耗开销进行控制. 该散列函数基于异或操作,通过调整 ICache 排队网络模型的节点转换概率,降低了共享 L1 ICache 的访问冲突. 实验结果表明,在指令缓存总容量为 32KB 的四核心簇上,使用 XOR 散列的共享 L1 ICache 结构较私有 L1 ICache 结构性性能平均优化 11%,较使用低位交错策略的共享 L1 ICache 结构性性能平均优化 8%,较使用面向跨步访存散列策略的共享 L1 ICache 结构性性能平均优化 3.2%.

关键词 单程序多数据流模型;指令缓存;众核处理器;排队网络模型;XOR 散列函数

中图法分类号 TP302 **DOI 号** 10.11897/SP.J.1016.2019.02499

Research and Design of XOR-Hash Indexing for Shared Instruction Cache

LIU Xiao TANG Yong ZHENG Fang DING Ya-Jun

(Jiangnan Institute of Computing Technology, Wuxi, Jiangsu 214083)

Abstract Single program multiple data (SPMD) is a main execution mode in the high-performance computing domain. While processing the same program segment, each adjacent core's execution varies depends on the data it processes and its own control flow. Many-core processor has been widely used in high performance computing domain for its advantages in high peak performance, high calculated density and high energy efficiency. While ensuring the performance, many-core processor has put forward higher requirements on power and area cost for it incorporates more cores and larger scale logic into a single chip. The SPMD execution mode can be effectively utilized by sharing the L1 instruction cache across adjacent cores. The strain on the on-chip resources is also alleviated by using the shared instruction cache. However, the sharing structure has the negative impact on performance, which is caused by access conflicts in the shared instruction cache. In this paper, we first give a theoretical analysis on access conflicts of the shared instruction cache based on the queuing network. Rather than physically corresponding banks of the instruction cache to queuing nodes, we model the shared instruction cache according to the cores' instruction fetch pattern. Queueing network reflects the steady-state performance of system when time tends to infinity. However, the access frequencies on each instruction cache bank tend to be the same in

such a long period of time. In other words, queueing network on physical cache banks may not precisely reflect the intensive conflicts on each bank. The theoretical analysis can achieve more accurate characteristics of access in shared instruction cache by utilizing the model on cores' instruction fetch pattern. The model of shared instruction cache given in this paper is later verified by simulation results. Based on the causes of performance loss in the theoretical analysis, we then design an XOR-hash function to minimize access conflicts in the shared L1 instruction cache. In the design of XOR-hash function, we accelerate the search of the hash function by leveraging null space of the hash matrix to eliminate the hash functions with the same hash effects. Under the premise of getting better randomization hash effects in the linear space of hash function, search cost and hardware implementation complexity are also taken into consideration by controlling the timing and power overhead. To control the propagation delay of the hash function, the maximum stages of the XOR-gate are restricted. We also limit the maximum load on each driver feeding to the XOR-gate to control the power overhead. By adjusting transformation probabilities of the instruction cache queueing network, this XOR-based hash function can effectively reduce bank conflicts in the shared L1 instruction cache. Experimental results show that in a 4-core cluster with 32KB instruction cache in all, the XOR hash-indexing shared instruction cache yields an 11% performance improvement compared to the private instruction cache and yields an 8% performance improvement compared to the set-interleaving shared instruction cache. The XOR hash-indexing shared instruction cache yields an 3.2% performance improvement compared to the shared instruction cache using the hash function for stride-based access.

Keywords single program multiple data; instruction cache; many-core processor; queueing network; XOR-hash function

1 引 言

在高性能计算领域,许多应用具有 SPMD(Single Program Multiple Data)的运行特征^[1-2],应用的各个划分执行相同的程序段.同时,由于处理数据或控制流的不同,划分后的执行过程具有一定差异,各核心间的独立性要优于 SIMT(Single Instruction Multiple Threads)模式下的处理器.处理核心的自主运行更利于有效发挥程序的执行效率.

指令高速缓存系统(Instruction Cache, ICache)的性能对处理器的性能影响较大^[3],较高的命中率、较低的平均访问延迟和对流水线持续高效供给指令的能力,是其结构设计的最基本要求.目前商用多核处理器的 ICache 性能基本都能达到该目标,多核处理器 ICache 的未来设计中需要考虑片上资源受限带来的影响.众核处理器中,由于片上集成的核心众多、芯片逻辑规模庞大、芯片资源紧张,众核的 ICache 设计对性能、面积和功耗开销提出了更高的要求^[4].

为了提高片上资源的利用率,可针对 SPMD 工作模式下邻近核心指令流相近的特点,将邻近核心的 ICache 共享起来,从而减少指令缓存系统的硬件开销,节省取指带宽.共享指令缓存能较大减少 L1 ICache 缓存的面积开销,但共享结构同时也会带来指令缓存的访问冲突和冲突缺失问题,对性能产生较大不利影响^[5].访问冲突不均匀情况下性能的损失问题会更加严重,因此访问冲突是共享 L1 ICache 上需要解决的重要问题之一.目前,缓解共享内存资源的访问冲突主要有两种技术方式:共享缓存分区和地址映射优化.

共享缓存分区主要通过限定访存地址在缓存中的位置来实现分区功能.共享缓存分区技术一般需要操作系统的支持,对组相连度也有较高的要求.在硬件资源有限的情况下,使用共享缓存分区技术需要权衡硬件开销或软件拷贝开销的不利影响^[6].

地址映射优化技术通过优化多 bank 存储系统上物理地址的映射行为,将连续的访存请求离散到不同的 bank 上,从而提升访存并行度,降低 bank 冲突带来的性能、带宽损失.其中, XOR 函数因具有

硬件实现开销低、散列随机效果多的特点,在多种访问模式下的存储体中得到了广泛应用。目前,学术界对并行存储系统中 LLC 和主存的 XOR 散列函数进行了大量研究,但针对共享 L1 ICache 运行特性的散列函数研究较少。由于邻近核心在 SPMD 模式下具有指令流高度相似的特点,共享 ICache 的散列函数需要针对相应的访问模式和分支特点进行专门设计。同时,由于指令高速缓存是众核处理器面积的重要消耗者,所以往往需要限制片上缓存系统的管理机制复杂度。共享 ICache 的 XOR 散列函数设计还需要在保证性能的前提下对设计复杂度、功耗和面积进行权衡。

本文第 2 节介绍相关工作;第 3 节基于闭合 Jackson 网络对共享指令缓存进行抽象建模,并对共享指令缓存访问冲突进行理论分析,探讨通过均衡转移概率来降低访问冲突的可能性;第 4 节介绍异或(XOR)散列函数定义及基本性质,同时基于闭合网络分析的共享指令缓存特点,设计并优化共享指令缓存上的 XOR 散列索引函数;第 5 节在 GEM5 体系结构模拟器^[7]上进行比较实验,并对结果进行分析。

2 相关工作

共享缓存分区是缓解共享内存资源访问冲突的主要技术路线之一。文献[8-10]采用路分区技术,该技术将高速缓存按路进行划分,通过将物理地址映射到高速缓存路的特定子集来实现分区,具有实现简单的优点。但路分区的可用分区数受限于高速缓存组相连度,可扩展性较差。同时,路分区导致的关联度下降会对性能造成不利影响。

组分区技术^[11-12]方式可以避免因关联度降低带来的性能损失,该技术需要利用操作系统的虚存管理技术将进程的物理地址限定到固定的缓存组中。但软件方法在页着色^[13]过程中需要对内存进行拷贝,开销较大。

散列函数常用于解决并行存储系统的访问冲突问题,一般通过离散存储系统中数据的映射位置来减少访存所造成的 Bank 冲突,从而达到提升访存性能的目的。此类函数具有将频繁发生的索引模式均衡扩散的能力。前人对并行存储系统上的散列函数进行了广泛研究。Gao 在质数内存中引入孙子定理,利用步长和 Bank 数互质来避免跨步访存模式的访问冲突^[14],该技术一般要求内存由质数个 bank

组成。

Rau 对缓存上的二进制多项式除法原理进行了研究^[15],并依据此原理提出了解决跨步访存冲突的 XOR 散列函数。文献[16]中指出 XOR 散列算法对降低小容量数据缓存上的访问冲突尤其有效。

Vandierendonck 采用线性代数理论对 XOR 散列函数进行了分析与设计^[17-18],通过对零核空间的研究,在保证随机散列效果的同时,有效减少了散列空间的搜索开销。

文献[19]分析了通用图形处理器(GPGPU)类众核处理器在 SPM(Scratchpad Memory)上的非单位跨步访存等多种访问模式,并设计了可配置的 SPM 随机异或散列索引。为了获得更强的随机散列能力,文献[19]采用了更多位数的索引作为散列函数的输入,但散列函数的时序开销也会相应增加。

文献[20]则对核心流水线上的 XOR 散列索引进行研究,通过缓解分支目标缓存的冲突缺失,避免了多级分支目标缓存的硬件开销,有效提升了指令分支预测的准确度。

3 基于闭合杰克逊网络的众核共享 L1 ICache 分析

本节使用排队论对共享指令缓存进行分析。排队系统广泛应用于计算机、通信和制造领域的模型分析与系统改进,是有效的建模设计方法和性能分析方法。其基于排队论的共享 L1 ICache 访问过程分析可提取缓存运行的特点,并有助于定位指令缓存的性能瓶颈,为缓解访问冲突问题提供帮助。

3.1 闭合杰克逊(Jackson)网络简介

访问冲突问题是共享 L1 ICache 结构上影响性能的重要因素,通过简化抽象可以转化为顾客-服务站问题。排队网络是解决顾客-服务站问题的有效工具。根据网络内的顾客总数是否为常数可划分为开放排队网络和闭合排队网络。闭合杰克逊(Jackson)网络是闭合排队网络的一种,其定义如下:

(1) 一个排队网络有 k 个节点,节点 $i(i=1, 2, \dots, k)$ 上有 c_i 个服务台,每个服务台服务单个顾客的时间服从指数分布,平均为 $1/\mu_i$ 。单个节点可以被视为一个 G/M/c/ ∞ 排队系统,即客户到达服从任意分布,服务模式为马尔可夫过程,有 c 个服务台和无限个等待位的排队系统。

(2) 没有顾客进入网络,即任意节点上,顾客以速率 $\lambda_i=0$ 通过节点 i 进入网络。

(3) 当一个顾客在节点 i 的服务台上完成服务, 它以概率 $r_{i,j}$ 进入节点 j , 以概率 r_{i0} 离开排队网络。

(4) 对任意节点 i , 都满足 $r_{i0}=0$ (没有顾客离开网络)。

由 Little's law^[21-22] 可以得到算法 1, 即闭合杰克逊网络节点的系统规模求解方法。通过此算法可以得到每个节点上的平均系统规模 $L_i(N)$ 。其中, N 是系统的总客户规模, $W_i(N)$ 指的是一个包含 N 个顾客的排队网络中顾客每次访问在节点 i 上的平均消耗时间, μ_i 是节点 i 上服务台的服务速率, $L_i(N-1)$ 指的是一个包含 $N-1$ 个顾客的排队网络中节点 i 上的平均顾客个数, $\lambda_i(N)$ 指的是包含 N 个顾客的排队网络中节点 i 上的顾客到达速率, $D_i(N)$ 表示 N 个顾客的排队网络中顾客访问节点 i 的平均时间间隔。由下列算法 1 可知, 网络节点转移概率 $r_{i,j}$ 是决定每个节点的平均系统规模(顾客个数)的重要因素。

算法 1. 闭合杰克逊网络节点系统规模求解。

输入: $N, k, r_{i,j}, \mu_i$

输出: 每个节点上的平均系统规模 $L_i(N)$

1. 设置任意一个 $\lambda_i=1$, 可求出所有 $\lambda_i = \sum_{j=1}^k \lambda_j r_{j,i}$
2. 初始化 $L_i(0)=0$
3. FOR ($n=1; n \leq N; n++$) DO
4. FOR ($i=1; i \leq k; i++$) DO
5. $W_i(n) = (1 + L_i(n-1)) / \mu_i$
6. END FOR
7. $\lambda_i(n) = n / D_i(n) = n / \sum_{i=1}^k \lambda_i W_i(n)$
8. FOR ($i=1; i \leq k; i++$) DO
9. IF ($i \neq l$) THEN
10. $\lambda_i(n) = \lambda_i(n) \lambda_i$
11. END IF
12. $L_i(n) = \lambda_i(n) W_i(n)$
13. END FOR
14. END FOR
15. RETURN $L_i(N)$

3.2 共享 L1 指令缓存的访问冲突理论分析

下文将通过闭合网络对访问冲突情况进行分析。首先对共享指令缓存建立模型, 做如下假设:

(1) 由于每个基准测试程序执行指令数量众多, 因此当程序核心段的指令足够多时, 其共享指令缓存的访问可以看作对任意节点 i , 都符合网络外客户到达速率 $\lambda_i=0$, 节点 i 上客户离开网络概率 $r_{i0}=0$ 的条件。即此排队网络没有顾客进入和离开网络。

(2) 无论指令缓存是否发生缺失, 其服务时间都服从指数分布。

(3) 缓存的每一个体看作一个节点, 且每个缓存节点上的服务速率一致。

(4) 顾客在每个节点有两种状态, 分别是获得服务和滞留。

基于以上假设可以得到共享指令缓存的抽象模型。对于四个核心共享一个四体的指令缓存, 可以把它看作一个 8 个节点的闭合杰克逊网络。

排队网络一般反映的是时间趋于无穷大时, 系统在稳态下的性能指标。但时间足够长、趋于无穷大时, CPU 对各个高速缓存体的访问频次基本一致。假如对物理节点进行直接映射建模, 则无法反映一个时间段内访问集中于某一指令块的访存特性。因此, 本文基于 CPU 对共享 ICache 体的访问特性进行建模。闭合网络模型节点对应关系如图 1 所示, 其中 4 个缓存节点对应共享指令缓存, 每个缓存节点是一个 G/M/1/ p 排队系统, p 是此缓存节点的排队系统容量, 服务速率设为常量 μ , 这 4 个节点的顾客之和是共享指令缓存的总顾客规模。此 8 节点排队网络还包含 4 个各阶段处理器节点, 每个此类节点内处理器对 ICache 体的访问模式相近。由于每个核心都可能处于任一个 ICache 体访问阶段, 则每个各阶段处理器节点可看作一个 G/M/4/ q 排队系统。此类排队节点的服务速率一般快于缓存节点, 记各阶段处理器节点的服务速率 $\mu_i = m\mu (m > 1)$, 即缓存节点服务速率的 m 倍。由于 GEM5 体系结构模拟器中每个核心指令请求的不可能超出取指缓冲容量, 即指令请求顾客总数小于每个节点的等待位, 所以分析中可以使用闭合杰克逊 (Jackson) 网络的计算公式。

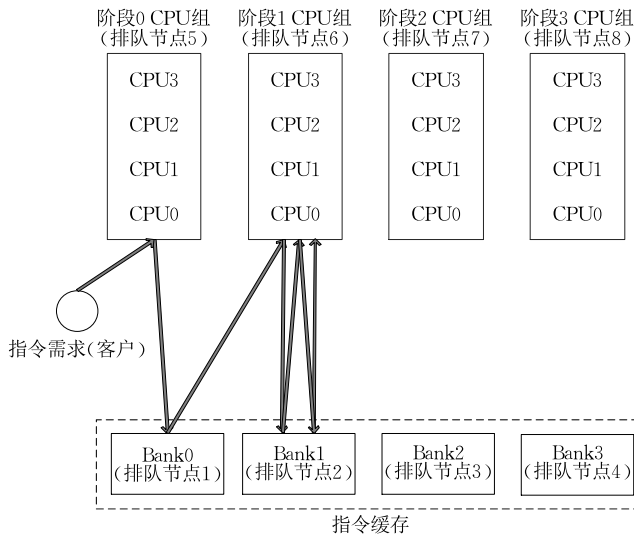


图 1 闭合网络模型节点对应关系

理论分析中将四个核心共享一个四体的指令缓存看作一个 8 个节点的闭合杰克逊网络. 表 1 是共享指令缓存排队网络转移概率的一般化表示, 节点转移概率 $r_{i,j}$ 在这里指的是指令作为“顾客”被完成“服务”后从节点 i 到节点 j 的概率. 其中节点 1~4 对应共享指令缓存各个体, 节点 5~8 是各个阶段的处理器节点. 顾客可以在各阶段处理器节点与缓存节点之间转移, 各阶段处理器分别访问特定的缓存体. 通过改变 $t_{i,j}$ 的值可以对转移概率值 $r_{i,j}$ 进行调整, 对不同特性的访问模式进行模拟.

表 1 四核心共享指令缓存排队网络转移概率			
i, j	$5 > i > 0, 8 \geq j \geq 5$	$8 \geq i \geq 5, j = i - 4$	其余情况
$r_{i,j}$	$t_{i,j} / (t_{i,5} + t_{i,6} + t_{i,7} + t_{i,8})$	1	0

上文对共享指令缓存初步建立模型, 接下来将通过部分 SPEC2006 基准对理论模型有效性进行验

证. 本文通过 GEM5 模拟的方式(详细实验设置见第 3 节)测试了课题的共享指令访问等待情况. 在本文假设下, 由于顾客总数小于每个节点的等待位, 分析中可以使用闭合杰克逊(Jackson)网络的计算公式.

3.2.1 共享指令缓存访问理论分析的模拟验证

表 2 是 SPEC2006 的浮点测试程序 leslie3d、libquantum 和整型测试程序 milc、h264ref 的共享指令缓存的转移比率值, 这里我们只统计缓存节点至各阶段处理器节点的转换率, 即 $5 > i > 0, 8 \geq j \geq 5$ 时的 $r_{i,j}$. 由表 2 中的程序转移比率值可知, 各个节点之间的转换比率不均匀. 其中, $j = i + 4$ 和 $j = ((i) \bmod 4) + 5$ 时转移比率 $r_{i,j}$ 的值一般较高, 两者之和平均约有 65%, 这是因为测试程序中顺序指令比例高, 处理器的相邻取指操作访问同一指令行或相邻指令行的概率较大. 其余两种转化比率低, 且差值较大.

表 2 测试程序 h264f、leslie3d、milc、libquantum 转移比率值																
测试程序	$r_{1,5}$	$r_{1,6}$	$r_{1,7}$	$r_{1,8}$	$r_{2,5}$	$r_{2,6}$	$r_{2,7}$	$r_{2,8}$	$r_{3,5}$	$r_{3,6}$	$r_{3,7}$	$r_{3,8}$	$r_{4,5}$	$r_{4,6}$	$r_{4,7}$	$r_{4,8}$
h264ref	0.382	0.358	0.104	0.156	0.078	0.415	0.453	0.054	0.107	0.273	0.259	0.360	0.413	0.045	0.203	0.339
leslie3d	0.470	0.263	0.002	0.265	0.064	0.476	0.460	0.001	0.036	0.155	0.509	0.300	0.479	0.041	0.203	0.277
milc	0.452	0.300	0.050	0.198	0.100	0.532	0.338	0.030	0.024	0.079	0.469	0.428	0.343	0.452	0.300	0.050
libquantum	0.846	0.006	0.003	0.146	0.001	0.023	0.975	0.001	0.001	0.955	0.019	0.025	0.099	0.015	0.010	0.875

由表 2 可计算出相应的各节点队列长度, 本文记各节点上理论客户规模为 L_i , 则缓存体总客户规模为 $\sum_1^4 L_i$. 如图 2 测试程序指令缓存理论队列比例, 这幅图中的横坐标是四个指令缓存体的编号, 纵坐标是各缓存体 L_i 占缓存体总客户规模 $\sum_1^4 L_i$ 的比例. 其中, 测试程序 libquantum 的冲突比例分布最不均匀, 其中, 比例最大差值达到 45%. 此共享指令缓存的访问冲突集中于 Bank3, 而 Bank1 和 Bank2 的利用率较低, 将节点转移比例均匀化有助于提升处理器性能.

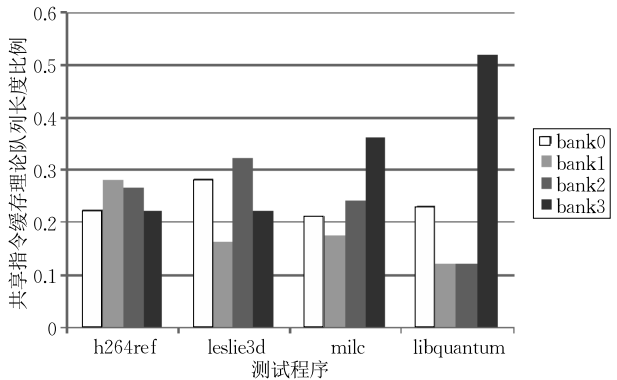


图 2 共享指令缓存理论队列比例

文中将缓存体访问冲突次数与指令缓存上访问冲突总次数的比率记为 $rate_{conflict}$. 图 3 是测试程序 h264ref、leslie3d、milc 和 libquantum 分别在四个核心共享一个四体的 ICache 结构下的指令缓存访问冲突次数比例, 横坐标是四个指令缓存体的编号, 纵坐标是各缓存体上的 $rate_{conflict}$. 其中, h264ref、milc 程序的访问冲突分布较为均匀; 程序 libquantum 的冲突分布最不均匀, 冲突比例最大差值高达 50%.

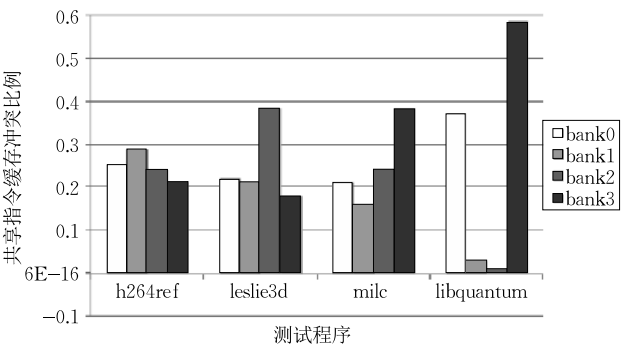


图 3 共享指令缓存访问冲突比例

对比图 2 指令缓存理论队列比例和图 3 指令缓存访问冲突比例可知, 共享 ICache 节点幅度 $L_i / \sum_1^4 L_i$ 和对应的访问冲突比例幅度基本一致, 即上述实际

测试基本验证了本文中共享指令缓存理论模型的有效性。

图 4 是部分 SPEC2006 测试程序运行时间比值,纵坐标是四核心共享 32KB L1 ICache 结构和每核心私有 8KB L1 ICache 结构在相同测试程序下的运行时间比值.可以看出,calculix、milc、soplex 程序受指令缓存访问冲突影响较小;程序 libquantum 因为指令访问冲突影响,性能损失明显.进一步证实了

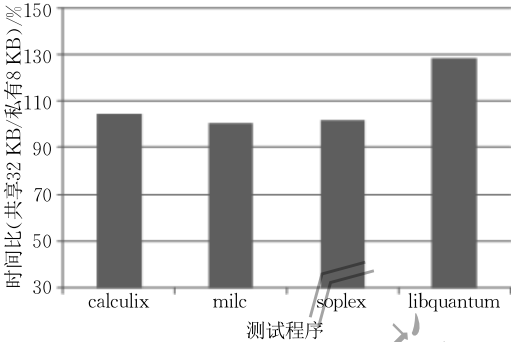


图 4 测试程序运行时间比值

访问冲突是共享缓存性能损失的重要原因之一。

接下来对不同指令访问模式下的共享指令访问等待情况进行分析。

3.2.2 低位交错共享指令缓存访问冲突理论分析

下面对不同转移概率下共享 ICache 的总访问规模进行探讨,即对指令节点顾客总规模进行探讨.低位交错是缓存常用的一种缓存块放置策略.本文先对这种缓存块放置策略做理论分析.由于各个缓存节点具有对称性,又根据程序中顺序指令比例大的特点,不失一般性假设指令缓存体 0 访问压力最大,指令缓存体 1 访问压力第二大,指令缓存体 2、3 访问压力最小.转移频次值设置如表 3 所示,分别设转移频次 $t_{i,j}$ 为 a_i^k 、 b_i^k 、 c_i^k 和 d_i^k ,通过改变指数 k 值和转移频次模拟值 a_i 、 b_i 、 c_i 、 d_i 可以对访问模式进行调整.此例子中转移频次模拟值 a_i 、 b_i 、 c_i 、 d_i 设置如表 4 所示,在此设置下随着指数 k 值的增加,转移概率值的差距越来越大,各个缓存节点的访问不均匀情况加剧。

表 3 转移频次 $t_{i,j}$ 值设置

i, j	$5 > i > 0, j = 5$	$5 > i > 0, j = 6$	$5 > i > 0, j = 7$	$5 > i > 0, j = 8$	$8 \geq i \geq 5, j = i - 4$	其余情况
$t_{i,5}$	$t_{i,5} / (t_{i,5} + t_{i,6} + t_{i,7} + t_{i,8})$	$t_{i,6} / (t_{i,5} + t_{i,6} + t_{i,7} + t_{i,8})$	$t_{i,7} / (t_{i,5} + t_{i,6} + t_{i,7} + t_{i,8})$	$t_{i,8} / (t_{i,5} + t_{i,6} + t_{i,7} + t_{i,8})$	1	0
$r_{i,j}$	$a_i^k / (a_i^k + b_i^k + c_i^k + d_i^k)$	$b_i^k / (a_i^k + b_i^k + c_i^k + d_i^k)$	$c_i^k / (a_i^k + b_i^k + c_i^k + d_i^k)$	$d_i^k / (a_i^k + b_i^k + c_i^k + d_i^k)$	1	0

表 4 转移频次模拟值 a_i 、 b_i 、 c_i 、 d_i 值

i	a_i	b_i	c_i	d_i
1	1.4	1.4	1.4	1.4
2	1.2	1.4	1	1
3	1	1	1.4	1
4	1	1	1	1.4

图 5 是低位交错共享指令缓存中理论访问比例,纵坐标分别代表特定 k 值时共享指令缓存各个体上的 $L_i / \sum_1^4 L_i$. 图中可以看出,随着指数 k 值的增加,共享指令缓存各个体的访问不均匀情况加剧.当 $k=1$ 时,各个体访问较均匀,其中 Bank0 访问比例最高,占 31%,Bank2、Bank3 访问比例最低,均只有

20%. 指数 $k=12$ 时访问比例最不均匀,最大差值达到 92%,其中 Bank0 访问比例最高,占 94%,Bank2、Bank3 访问比例最低,均只有 2%。

图 6 是四处理器共享低位交错指令缓存排队网络中指令缓存节点排队系统的总指令行“顾客”规模,由图中可以看出,共享指令缓存总访问规模 $\sum_1^4 L_i$ 会随着指数 k 值的增加而增长.当 $k=1$ 时共享指令缓存各个体的访问分布最均匀,缓存总指令行规模(因为访问冲突被阻塞的指令请求数)只有 6.11. 当共享指令缓存访问分布最不均匀时,指令缓存节点总指令请求“顾客”数达到 6.85,访问冲突对

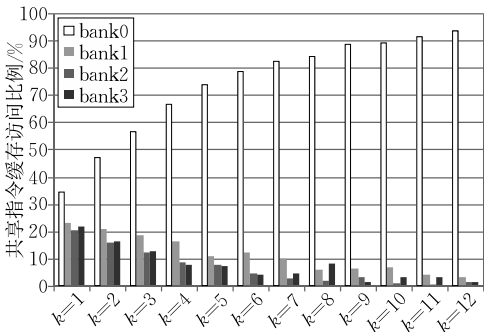


图 5 低位交错共享指令缓存理论访问比例

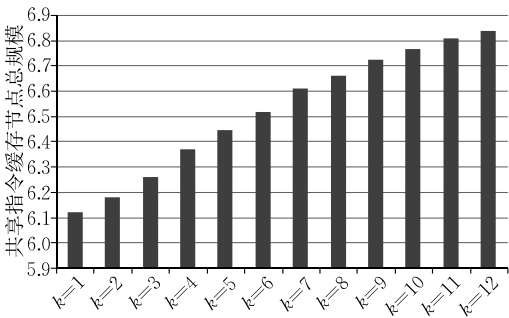


图 6 低位交错共享指令缓存总访问数

处理器总体性能的不利影响最大。

综上所述, $r_{i,j}$ 节点转换概率可以对共享指令缓存的指令行规模造成影响。当处理器访问各指令缓存体均匀(节点转换概率 $r_{i,j}$ 合理)时, 共享指令缓存利用率高, 缓存上访问冲突对处理器性能的不利影响较小。

4 众核共享 L1 ICache 散列函数算法设计

由上一节分析可知, 共享指令缓存上 $r_{i,j}$ 节点转换概率可以影响指令缓存访问冲突次数。将各指令行散列到缓存的各个体中, 可以均衡 $r_{i,j}$ 节点转换概率的值, 从而降低共享指令缓存上的访问冲突。本节将对适应共享指令缓存的散列函数进行研究。

4.1 XOR 散列函数

本文将根据所提出的指令缓存系统特点, 对散列函数算法进行研究。

定义 1. 当 p 为素数时, $F = \{0, 1, 2, \dots, p-1\}$ 在 $\text{mod}(p)$ 下关于模运算的加法和乘法构成一个有限群, 这个群就记为 $GF(p)$ 。

计算机领域, 通常使用在 $GF(2)$ 上的散列函数。此时, 地址域 $\{0, \dots, 2^{n-1}\}$ 可以由向量 $\mathbf{a} = [a_{n-1} \dots a_1 a_0]$ 表示, 索引域 $\{0, \dots, 2^{m-1}\}$ 可以由向量 $\mathbf{z} = [z_{m-1} \dots z_1 z_0]$ 表示, 这里 $n > m > 0$ 。索引向量 $\mathbf{z}$ 的计算可以表示为 $\mathbf{f}\mathbf{a} = \mathbf{z}$ 。XOR 散列函数 f 实现简单, 其加法和乘法分别对应异或(XOR)操作和与(AND)操作, 本文主要针对 XOR 散列函数进行设计。索引值计算可以用一个二值变换矩阵 $\mathbf{H}$ 来表示:

$$\mathbf{H} = \begin{bmatrix} h_{n-1,m-1} & \cdots & h_{n-1,0} \\ \vdots & \ddots & \vdots \\ h_{0,m-1} & \cdots & h_{0,0} \end{bmatrix}.$$

上式 $\mathbf{H}$ 中 $h_{i,j}$ 为 1 时, a_i 作为 b_j 计算路径上 XOR 门的一个输入。本文中把矩阵 $\mathbf{A}$ 的秩记为 $R(\mathbf{A})$, $R(\mathbf{H}) = m$ 。

$$0 \leq R(\mathbf{A}_{m \times n}) \leq \min(m, n) \quad \text{性质 1}$$

$$R(\mathbf{AB}) \leq \min\{R(\mathbf{A}), R(\mathbf{B})\} \quad \text{性质 2}$$

$$n \times m \text{ 矩阵 } \mathbf{A} \text{ 列满秩}(n > m) \Leftrightarrow$$

$$\mathbf{A} \text{ 的标准型为 } \begin{pmatrix} \mathbf{E}_m \\ \mathbf{0} \end{pmatrix} \quad \text{性质 3}$$

若 $n \times m$ 的矩阵 $\mathbf{A}$ 的秩 $R(\mathbf{A}) = r$, 则 n 元齐次线性方程组 $\mathbf{x}\mathbf{A} = \mathbf{0}$ 的解集 S 的秩 $R = n - r$ 性质 4
易知 $\mathbf{H}$ 的表示有 2^{mn} 种可能, $\mathbf{H}$ 的设计空间过大导致

致较长的搜索时间, 不利于散列函数的设计。虽然一些地址域经过映射之后会在不同的组发生冲突, 但这些映射可能有相同的散列效果。Vandierendonck 指出这部分映射对应的矩阵可能有相同的零核空间(Null Space), 记为

$$N(\mathbf{H}) = \{\mathbf{x} \mid \mathbf{x}\mathbf{H} = \mathbf{0}, \mathbf{x} \in \{0, 1\}^{1 \times n}\} \quad (1)$$

$$\mathbf{a}\mathbf{H} \oplus \mathbf{b}\mathbf{H} = \mathbf{0} \Leftrightarrow (\mathbf{a} \oplus \mathbf{b})\mathbf{H} = \mathbf{0} \Leftrightarrow \mathbf{a} \oplus \mathbf{b} \in N(\mathbf{H}) \quad (2)$$

当 $\mathbf{a}\mathbf{H} \oplus \mathbf{b}\mathbf{H} = \mathbf{0}$ 时, 同时访问地址 a 与 b 时会产生访问冲突。由式(2)可知, $\mathbf{a} \oplus \mathbf{b}$ 是 $N(\mathbf{H})$ 中的向量之一。零核空间能把散列函数变换矩阵 $\mathbf{H}$ 划分为一系列的等价类。 $\mathbf{H}$ 映射矩阵 $\mathbf{H}$ 秩为 m , 由性质 4 可知 $N(\mathbf{H})$ 的秩为 $n - m$, 即共有 2^{n-m} 个无关向量。所以, 当增大 $n - m$ 时零核空间的种类可以指数增长, 相应地得到更多不同随机效果的散列函数, 增加了找到更优随机效果函数的可能性。

4.2 共享指令缓存散列索引函数设计

XOR 散列函数在非可重构情况下可以只使用 XOR 操作完成散列变换, 这种散列函数具有实现简单、附加延迟低的特点。 $\mathbf{H}$ 的线性空间内存在多个 XOR 散列函数, 每个散列函数的散列特性差异较大, 通过在线性空间内寻找最合适的散列函数矩阵可以优化 XOR 散列函数。

本文以 64 字节长度指令行、四个体直接映射的 32KB 共享指令缓存为例, 取 $PC[13:7]$ 作为散列函数输入值生成 2 位的索引值。上述计算方法对应的变换矩阵 $\mathbf{H}_{16 \times 2}$ 解空间较大, 有 2^{14} 种可能的矩阵表示方式。基于对计算代价和工程实现的考虑, 可以对 $\mathbf{H}$ 的搜索中作如下限制, 以下约束均可以减小 $\mathbf{H}$ 解空间的范围:

(1) 通过限制 AND 和 XOR 门的级数可以达到降低延迟的目的。本文中将散列变换电路的门延迟控制在 2 级以下, 即 $\mathbf{H}$ 矩阵 0 到 $m-1$ 列中, 需要将每列 1 的个数控制在 4 以下。

(2) 输入信号扇出的增长需要增加相应驱动, 带来功耗开销。因此, 本文中将信号扇出值限制在 4 以下。在变换矩阵 $\mathbf{H}$ 中体现为: m 到 $n-1$ 行, 每行所含 1 的个数不超过 4。在本文选择的设置下 $\mathbf{H}$ 设计空间内所有矩阵均符合这一要求。

(3) 为了避免进一步的访问冲突, 矩阵 $\mathbf{H}$ 应保证连续的 2^m 个连续向量映射到不同的组中, 则变换矩阵 $\mathbf{H}$ 低 m 行中每行至少包含一个 1。又由矩阵的初等列变化不影响其列向量空间易知 $\mathbf{H}$ 列满秩, 即 $\mathbf{H}$ 列向量线性无关, 则矩阵 $\mathbf{H}$ 可以通过初等列变换

将低 m 行变成对角阵,在本文的设置下相应的矩阵 $\boldsymbol{H}$ 的设计空间由 2^{14} 减小为 2^{10} .

图 7 是低位交错索引和部分符合约束的 XOR 散列函数对应的变换矩阵, $\boldsymbol{H}_{\text{传统}}$ 指的是低位交错放置方式对应矩阵. 由于符合约束 XOR 散列函数有多个,需要对这部分函数进行筛选. 本文选择访存冲突次数作为评价标准,由于访问冲突会影响众核簇的动态运行行为,矩阵 $\boldsymbol{H}$ 的访问冲突次数统计需要在实际运行测试程序时进行.

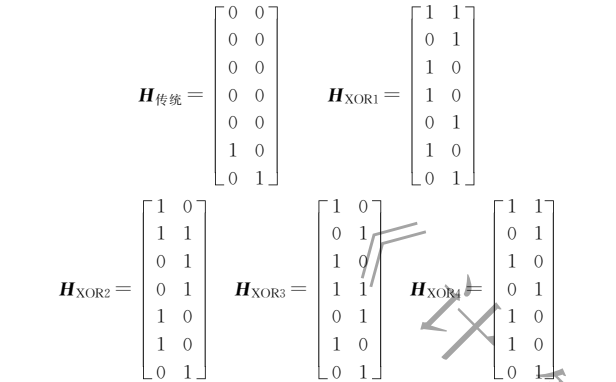


图 7 低位交错索引和 XOR 散列索引的变换矩阵

低位交错是缓存常用的一种缓存块放置策略,低位交错和散列映射矩阵 $\boldsymbol{H}_{\text{XOR4}}$ 的映射方式如图 8 所示,可以看出相比图 8(a)低位交错映射方式,图 8(b)XOR 散列能更随机地将指令行映射至共享缓存的各个体中.

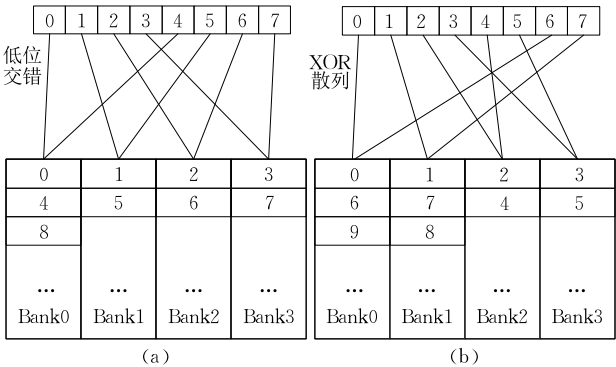


图 8 低位交错索引和 XOR 索引

由于各个测试程序的指令流特点具有差异性,每个测试程序上的较优 XOR 散列函数不同. 本文基于访问冲突次数对散列索引进行选取,综合考虑 XOR 索引在多个测试程序上的优化结果,最终选取变换矩阵 $\boldsymbol{H}_{\text{XOR4}}$ 作为本文的 XOR 索引函数. 基于访问冲突次数的比较可以获得较好的变换矩阵 $\boldsymbol{H}_{\text{XOR4}}$,该矩阵对应的逻辑电路如图 9 所示,该逻辑具有面积开销小,延迟低的特点.

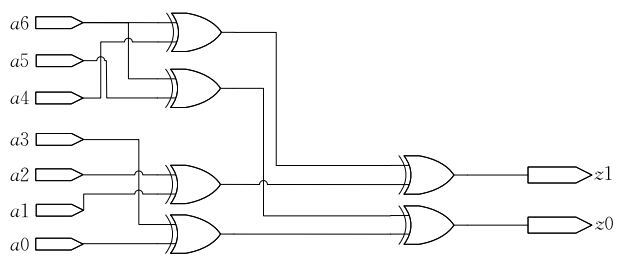


图 9 $\boldsymbol{H}_{\text{XOR4}}$ 的逻辑电路图

5 实验结果及分析

本文基于 GEM5 模拟器对共享指令缓存进行二次开发和模拟,同时利用 GEM5 中的指标统计器进行测试数据的统计. 各配置下 ICache 的面积评估基于惠普实验室的 CACTI^[23] 工具. 本文对不同指令缓存结构下的处理器性能进行了对比和分析,测试中所使用的 SPEC2006 测试程序均使用 gcc3. 4-o3 选项进行编译. 处理器相关参数如表 5 所示,其余参数同 GEM5 默认设置. 实验每次选择共享 L1 ICache、私有 8 KB L1 ICache、私有 32 KB L1 ICache 中的一种指令缓存对处理器进行配置. 模拟中对这几种结构进行性能测试,并评估文中设计的共享 L1 ICache 散列索引函数.

表 5 模拟配置

结构	配置参数
指令体系结构	ALPHA
核心配置	2GHz 8 取指 2 发射 O3 乱序处理器模型 取指缓冲 64 B
共享 L1 ICache	32 KB 共享指令缓存共 4 个体 512 条指令行 直接映射 命中延迟 2 周期
私有 8 KB L1 ICache	8 KB 指令缓存共 1 个体 128 条指令行 直接映射 命中延迟 2 周期
私有 32 KB L1 ICache	32 KB 共享指令缓存共 1 个体 512 条指令行 直接映射命中延迟 2 周期
L1 DCache	每核心 32 KB 私有数据缓存 512 条指令行 两路组相连 命中延迟 2 周期
L2 Cache	8 MB 共享混合缓存 八路组相连 命中延迟 40 周期
指令行长度	64 B(16 条指令)
目标工艺	32 nm

5.1 测试程序动态指标分析

本文首先对 8 KB 私有指令缓存处理器的动态指标进行分析. 8 KB 私有指令缓存处理器的性能计

数器结果如表 6 所示,表中 L2 指令缺失率是 L2 缓存中指令总缺失数和 L2 缓存中总指令请求数的比值. 观察平均顺序指令块长度可知,libquantum 和 omnetpp 的顺序指令块长度最小,不到 6 条指令便有一个跳转;测试程序 milc 的平均顺序指令长度最长,同一顺序指令块跨指令行的概率较大. 本文配置下所选程序的 L2 指令缺失率普遍较低,其中只有测试程序 leslie3d 的 L2 指令缺失率较大. L2 缓存已满足大部分应用的性能要求,并不是性能瓶颈. L1 指令缓存的平均缺失率为 8%,测试程序 namd、omnetpp 和 soplex 的 L1 指令缓存缺失率较大,分别是 10%、34%和 21%. 测试程序 gobmk 和 hmmer 的 L1 指令缺失率最小,分别是 0.03%和 0.023%.

表 6 SPEC2006 统计数据(8 KB 私有 L1 ICache)			
SPEC2006	平均顺序指令块长度	L2 指令缺失率	L1 指令缺失率
calculix	7.34	0.001095	0.063092
bzip	8.90	0.001674	0.006978
gromacs	21.56	0.000292	0.055676
h264ref	13.28	0.001898	0.091432
leslie3d	27.88	0.548158	0.026012
milc	30.28	0.005565	0.041849
namd	10.93	0.000015	0.100884
omnetpp	5.29	0.000024	0.343888
soplex	6.96	0.004516	0.212721
gobmk	5.95	0.027302	0.000302
libquantum	5.11	0.010887	0.000239
hmmer	14.32	0.002556	0.027440

5.2 各配置下 L1 ICache 的面积评估

如表 7,本文基于惠普实验室的 CACTI 工具,对各配置下 ICache 的面积进行了评估. 其中共享 32 KB ICache 指的是多核心共享 32 KB 的指令缓存,该共享 ICache 可以采用低位交错映射策略或 XOR 散列映射策略. 四核心配置下,私有 ICache 结构每个核心配置一个私有 ICache,即私有 ICache 总面积开销是四个私有 ICache 面积之和. 四核心配置下共享 32 KB ICache 由四个核心共享,ICache 总面积开销为一个共享 32 KB ICache 的面积.

表 7 L1 ICache 面积评估				
ICache 配置	bank 数	数据阵列面积/mm ²	Tag 阵列面积/mm ²	四核心配置下 ICache 总面积/mm ²
私有 32 KB ICache	1	0.03050570	0.00400998	0.13806272
共享 32 KB ICache	4	0.04841640	0.00623978	0.05465618
私有 8 KB ICache	1	0.00921198	0.00129655	0.04203412

由表 7 的数据分析可知,四核心共享 32 KB ICache 较四核心私有 8 KB ICache 的总指令缓存面积增加 30%,四核心私有 32 KB ICache 较四核心私有 8 KB ICache 的总指令缓存面积增加 228.45%.

5.3 共享 L1 ICache 冲突优化效果及分析

本文在 GEM5 模拟器上对使用散列索引 H_{XOR4} 前后的共享 L1 ICache 结构进行了测试,并对这两种情况下的访问冲突情况进行了分析. 本文将缓存体上访问冲突次数与 ICache 访问冲突总次数的比率记为 $rate_{conflict}$. 图 10 是按低位交错放置指令行的共享 ICache 中各缓存体冲突比例,Bank 0~3 分别代表特定测试程序下各个体上的 $rate_{conflict}$. 可以观察到各个测试程序的冲突分布情况有所不同,其中测试程序 h264ref、bzip2 的冲突分布较为均匀,而 libquantum、gobmk 和 hmmer 分布较不均匀,冲突比例差最大能达到 55%,这种情况下访问冲突对处理器性能影响较大.

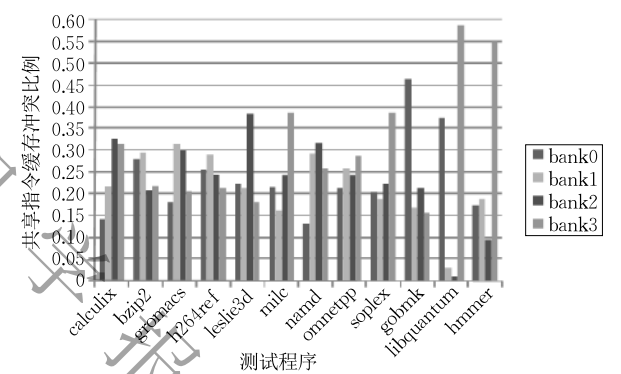


图 10 低位交错共享指令缓存各缓存体冲突比例

图 11 是使用 H_{XOR4} 散列函数的共享 ICache 各缓存体冲突比例,纵坐标是共享 ICache 各缓存体上冲突比例. 观察图中数据可以看出,大部分测试程序的冲突不均匀情况在使用散列函数后有一定缓解. 测试程序 libquantum、hmmer 和 namd 的最大冲突比例差降低了 15%、16%和 19%,而测试程序 milc

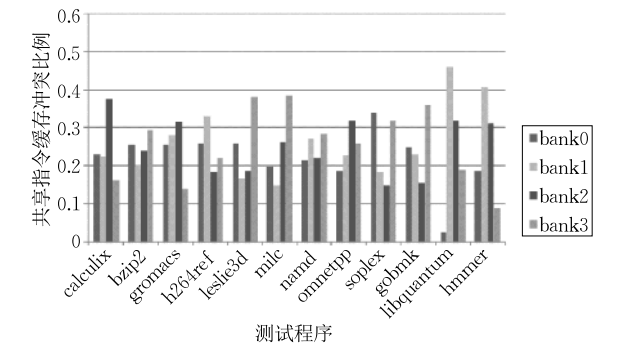


图 11 散列共享指令缓存各缓存体冲突比例

的冲突不均匀情况基本没有变化. XORHASH 结构较低位交错结构均方差平均降低约 5.3%,这说明 H_{XOR4} 散列共享 ICache 各个体的访问更加均匀,每个体较低位交错共享 ICache 的每个体利用率更高.

5.4 共享 L1 ICache 散列索引性能优化效果及分析

图 12 是两种共享 ICache 结构和私有 ICache 结构的性能比较结果. 其中私有 32 KB L1 指的是四核心分别拥有 32 KB L1 ICache 的处理器结构. 在 L2 指令缺失率较低的情况下,该结构能达到的性能可认为是共享 32 KB 指令缓存的加速上限;低位交错指的是四核心共享 32 KB 低位交错共享 ICache 的处理器结构;XOR_STRIDE 是四核心共享 32 KB ICache 的处理器结构,该结构采用文献[8]中面向跨步访存冲突的 XOR 散列函数;XORHASH 指的是四核心共享 32 KB 散列映射 H_{XOR4} 的共享 ICache 的处理器结构. 图 12 的纵坐标是这几种结构和四核心私有 8 KB L1 ICache 的处理器结构在运行对应测试程序的时间比值.

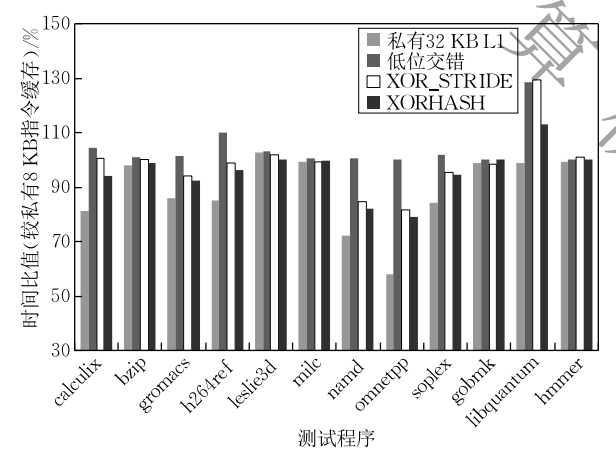


图 12 不同指令缓存运行时间比值

由表 6 的数据分析可知,测试程序 h264ref、namd、omnetpp、soplex 通过增加指令缓存容量后性能提升的潜力较大,gobmk、libquantum 程序的 L1 指令缺失率最小,在使用共享 ICache 时可能会因访问冲突遭受性能损失,图 12 中的结果验证了表 6 的数据分析. 平均来看,私有 32 KB 结构、低位交错结构、XOR_STRIDE 结构和 XORHASH 结构分别较 8 KB 私有 ICache 性能平均优化 15%、3%、7.5% 和 11%.

可以观察到,在核心私有 L1 ICache 容量扩大到 32 KB 以后,各测试程序性能都有不同程度的提升. h264ref、namd、omnetpp 性能提升最明显,运行时间减少了约有 20%;gobmk、libquantum 程序的性能基本没有提升,这部分结果与表 6 的数据分析

相符合.

而观察图 12 中低位交错结构的数据可以发现,此结构在运行测试程序 namd、omnetpp 性能提升较多,约有 5%. 但由于受到访问冲突的影响,低位交错结构在运行其余测试程序时性能较 8 KB 私有 ICache 结构没有提高,libquantum、h264ref 测试程序对应的性能甚至有所降低.

XOR_STRIDE 散列面向跨步访存模式进行设计,较 8 KB 私有 ICache 性能平均优化 7.5%,较低位交错结构性能平均优化 4.6%. 但在测试程序 libquantum 上,由于面向跨步的映射模式加剧了指令缓存访存冲突,XOR_STRIDE 结构较低位交错结构性能恶化 1%.

XOR 散列结构的指令缓存面积为私有 32 KB L1 ICache 的 39%,节省了较多的面积开销. XOR 散列结构虽然也受访问冲突的影响,性能劣于私有 32 KB L1 ICache 结构约 5%,但其性能较 8 KB 私有 ICache 结构有所提升,平均优化 11%. 其中,对 L1 ICache 容量要求较高的测试程序 namd、omnetpp 性能提升最多,XOR 散列结构较 8 KB 私有 L1 ICache 结构提升了约 20%.

低位交错、XOR_STRIDE 和 XORHASH 均是四个核心共享 32 KB L1 ICache 的结构,XORHASH 结构由于访问冲突更加均匀,性能总体上优于低位交错,约优化 8%. 在使用 XORHASH 后,较低位交错结构中性能损失较大的测试程序 libquantum 运行时间减少了 13%. XORHASH 结构面向指令访存模式进行设计,较面向跨步冲突的 XOR_STRIDE 结构性能平均优化 3.2%.

5.5 共享 L1 ICache 容量及核心数目的性能影响

另外,本文对不同配置情况下的共享 ICache 结构性能进行了测试. 综合考虑性能和面积开销因素,四核心共享 32 KB L1 ICache 结构在测试中表现较优.

图 13 是四核心处理器使用不同共享 ICache 下的时间比值,共享 16 KB~64 KB 分别代表 16 KB 到 64 KB 使用低位交错放置策略的共享指令缓存,XOR 共享 16 KB~64 KB 结构代表采用 XOR 散列放置策略的 16 KB 到 64 KB 共享指令缓存. 纵坐标是相同测试激励下各指令缓存结构和共享 32 KB 结构的运行时间比值,横坐标是相应测试激励.

不同测试激励下各结构性能均受到 ICache 容量大小和 ICache 访问冲突的影响,放置策略相同时处理器性能随共享指令缓存容量地增加而提升. 从

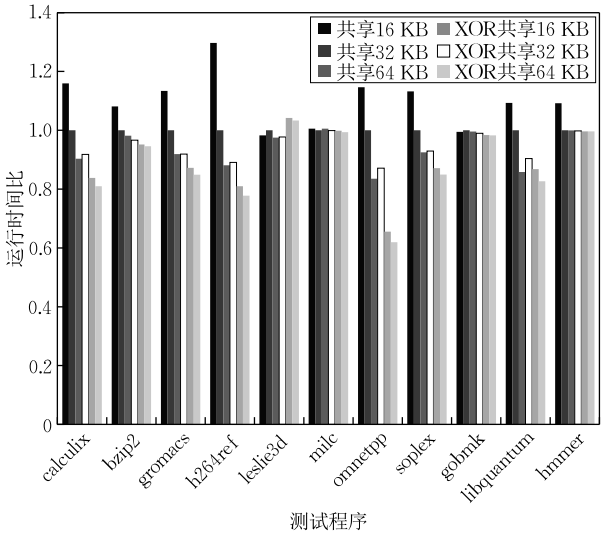


图 13 4 核心各结构共享指令缓存性能比较

图 13 中可以看出,当共享 ICache 容量相同时,采用 XOR 散列放置策略结构的性能一般要优于采用低位交错策略结构的性能,平均运行时间优化约有 11%。其中,测试激励 calculix、gromacs、gobmk 下 XOR 共享 16 KB 结构与共享 64 KB 结构性能持平。测试程序 mile 下各结构性能相差不大。测试激励 omnetpp 下 XOR 结构性能优化效果最明显,相同容量下较共享 16 KB~64 KB 结构性能分别优化 27.5%、34.5%、18%。

图 14 是 2 至 8 核心使用不同放置策略共享 ICache 的运行时间比值。图中纵坐标是对应测试激励下各指令缓存结构和四核心共享 32 KB 结构的运行时间比值,2~8 核心 XOR 指的是 2 至 8 核心共享 32 KB 的 XOR 散列放置策略 ICache 结构,2~8 核心共享是 2 至 8 核心共享采用低位交错放置策略的 32 KB 指令缓存结构。

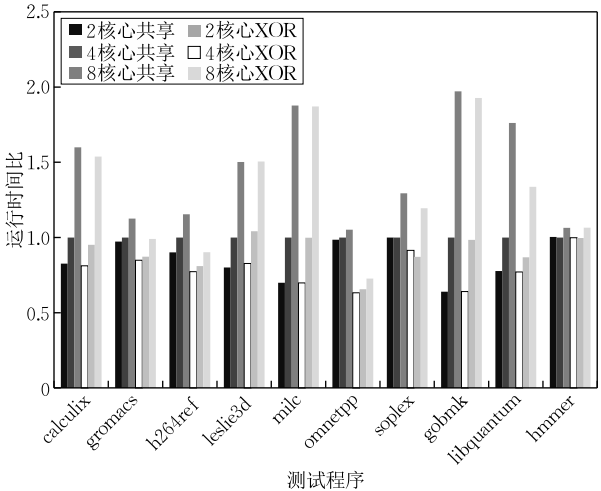


图 14 2~8 核心共享 32 KB 指令缓存性能比较

指令缓存容量和放置策略不变时,随着核心数目的增加,共享指令缓存访问冲突增多、性能降低。图 14 中 4 核心共享结构较 2 核心共享结构运行时间平均恶化幅度约有 4%,8 核心共享结构较 2 核心共享结构运行时间平均恶化幅度约有 43%。4 核心 XOR 结构较 2 核心 XOR 结构运行时间平均恶化幅度约有 4%,8 核心 XOR 结构较 2 核心 XOR 结构运行时间平均恶化幅度约有 34%。因此,建议采用四核心共享同一指令缓存的配置来降低指令缓存面积开销。

核心数目相同时,采用 XOR 散列放置策略共享指令缓存较低位交错策略共享指令缓存性能优化约 15%。其中,libquantum、omnetpp 测试激励下 XOR 散列放置策略优化效果明显。libquantum 测试激励下 2~8 核心 XOR 散列放置策略较低位交错放置策略分别优化 0.6%、14%、40%。omnetpp 测试激励下 2~8 核心 XOR 散列放置策略较低位交错放置策略分别优化 35%、34%、32%。

6 总 结

本文提出了基于 XOR 散列函数的共享指令缓存,此结构利用 SPMD 工作模式下相邻核心指令流相似的特点,通过共享邻近核心的 L1 ICache,可以达到减少指令缓存系统硬件开销、节省取指带宽的目的。本文基于闭合 Jackson 网络对共享指令缓存进行抽象建模,设计了面向共享 ICache 的低冲突 XOR 散列函数。该散列函数具有实现开销小、计算延迟低的特点,通过均匀共享 ICache 各个体的访问压力,有效提高了系统性能。实验表明,在核心数目为 4、指令缓存总容量为 32 KB 的情况下,使用 XOR 散列的共享 32 KB L1 ICache 结构较私有 8 KB L1 ICache 结构的运行时间平均优化 11%,较使用低位交错策略的共享 32 KB L1 ICache 结构运行时间平均优化 8%,较使用面向跨步访存散列策略的共享 32 KB L1 ICache 结构运行时间平均优化 3.2%。其次,相比商用处理器一般采用的私有 32 KB L1 ICache 结构,使用 XOR 散列的四核心共享 32 KB L1 ICache 结构能在较小性能损失的情况下将指令缓存面积开销降低 61%。最后,本文还对不同配置情况下的共享指令缓存结构性能进行了测试。综合考虑面积开销和性能影响,四核心共享 32 KB L1 ICache 结构的配置较为合适。

致 谢 非常感谢数学工程与先进计算国家重点实验室的张昆博士及各位共同作者在研究中提供的宝贵帮助,在此表示诚挚谢意!

参 考 文 献

- [1] Carter N P, Agrawal A, Borkar S, et al. Runnemed: An architecture for ubiquitous high-performance computing//Proceedings of the 19th International Symposium on High Performance Computer Architecture. Shenzhen, China, 2013: 198-209
- [2] Zheng Fang, Zhang Kun, Wu Gui-Ming, et al. Architecture techniques of many-core processor for energy-efficient in high performance computing. Chinese Journal of Computers, 2014, 37(10): 2176-2186(in Chinese)
(郑方, 张昆, 邬贵明等. 面向高性能计算的众核处理器结构级高效技术. 计算机学报, 2014, 37(10): 2176-2186)
- [3] Hayenga M, Naresh V R K, Lipasti M H. Revolver: Processor architecture for power efficient loop execution//Proceedings of the 20th International Symposium on High Performance Computer Architecture. Orlando, USA, 2014: 591-602
- [4] Molka D, Hackenberg D, Schöne R. Main memory and cache performance of intel sandy bridge and AMD bulldozer //Proceedings of the Workshop on Memory Systems Performance and Correctness. Edinburgh, UK, 2014: 4-14
- [5] Balasubramonian R, Jouppi N, Muralimanohar N. Multi-core cache hierarchies. Synthesis Lectures on Computer Architecture, 2011, 6(3): 1326-1327
- [6] Gao Ke, Chen Li-Cheng, Fan Dong-Rui, et al. Shared memory resources allocation and management research on multicore systems. Chinese Journal of Computers, 2015, 38(5): 1020-1033(in Chinese)
(高珂, 陈荔城, 范东睿等. 多核系统共享内存资源分配和管理研究. 计算机学报, 2015, 38(5): 1020-1033)
- [7] Binkert N, Beckmann B, Black G, et al. The GEM5 simulator. ACM SIGARCH Computer Architecture News, 2011, 39(2): 1-7
- [8] Suh G E, Devadas S, Rudolph L. A new memory monitoring scheme for memory-aware scheduling and partitioning//Proceedings of the 8th International Symposium on High-Performance Computer Architecture. Cambridge, USA, 2002: 117-128
- [9] Chiou D, Jain P, Devadas S, Rudolph L. Dynamic cache partitioning via columnization//Proceedings of the 37th Annual Design Automation Conference. Los Angeles, USA, 2000: 416-419
- [10] Cook H, Moreto M, Bird S, et al. A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness//Proceedings of the 40th Annual International Symposium on Computer Architecture. Tel Aviv, Israel, 2013: 308-319
- [11] Ranganathan P, Adve S, Jouppi N P. Reconfigurable caches and their application to media processing//Proceedings of the 27th International Symposium on Computer Architecture. Vancouver, Canada, 2000: 214-224
- [12] Varadarajan K, Nandy S K, Sharda V, et al. Molecular caches: A caching structure for dynamic creation of application-specific Heterogeneous cache regions//Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. Washington, USA, 2006: 433-442
- [13] Jiang L, Lu Q, Ding X, et al. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems//Proceedings of the 14th International Conference on High Performance Computer Architecture. Salt Lake City, USA, 2008: 367-378
- [14] Gao Q S. The Chinese remainder theorem and the prime memory system//Proceedings of the International Symposium on Computer Architecture. San Diego, USA, 1993: 337-340
- [15] Rau B R. Pseudo-randomly interleaved memory. ACM SIGARCH Computer Architecture News, 1991, 19(3): 74-83
- [16] Topham N, González A. Randomized cache placement for eliminating conflicts. IEEE Transactions on Computers, 1999, 48(2): 185-192
- [17] Vandierendonck H, De Bosschere K. XOR-based hash functions. IEEE Transactions on Computers, 2005, 54(7): 800-812
- [18] Vandierendonck H, Bosschere K D. Constructing optimal XOR-functions to minimize cache conflict misses. Architecture of Computing Systems, 2008, 4937: 261-272
- [19] Braak G J V D, Gomezluna J, Gonzaleslinares J, et al. Configurable XOR hash functions for banked scratchpad memories in GPUs. IEEE Transactions on Computers, 2016, 65(7): 2045-2058
- [20] Wang Guo-Peng, Hu Xiang-Dong, Yin Fei, et al. Research and design of hash indexing mechanism for BTB. Journal of Computer Research & Development, 2014, 51(9): 2003-2011(in Chinese)
(王国澎, 胡向东, 尹飞等. BTB 索引散列算法的研究与设计. 计算机研究与发展, 2014, 51(9): 2003-2011)
- [21] Allen A O. Probability, Statistics, and Queueing Theory. Salt Lake City, USA: Academic Press, 1978
- [22] Gross D, Shortle J F, Thompson J M, et al. Fundamentals of Queueing Theory. 4th Edition. Hoboken, USA: John Wiley & Sons, 2008
- [23] Muralimanohar N, Balasubramonian R, Jouppi N P. Architecting efficient interconnects for large caches with CACTI 6.0. IEEE Micro, 2008, 28(1): 69-79



LIU Xiao, M.S., assistant engineer. His research interests include high performance computing and computer architecture.

TANG Yong, M. S., assistant professor. His research interests include high performance computing and computer architecture.

ZHENG Fang, Ph. D., assistant professor. His research interests include high performance computing and computer architecture.

DING Ya-Jun, M. S., senior engineer. His research interests include high performance computing and computer architecture.

Background

This work focuses on XOR-hash indexing for the shared instruction cache of many-core processor in high performance computing.

Many-core processor has become the main development trend in high performance computing domain for its advantages in high peak performance, high calculated density and high energy efficiency. Single Program Multiple Data (SPMD) is a main execution mode of applications in high-performance computing domain. While processing the same program segment, every core’s execution varies depends on the data it processes and its control flow. It is important to improve the efficiency of instruction cache in many-core processor to achieve higher performance in SPMD execution mode. While ensuring the performance, many-core processor has put forward higher requirements of power and area cost for it incorporating more cores and larger scale logic into a single chip. In the aspect of area, it is essential to limit the complexity and capacity of on-chip cache because the instruction cache consumes a part of the area and power of many-core processor.

There exist several previous works focus on power and performance of instruction cache system between L1 instruction cache to front-end of pipeline. For example, micro-op cache, loop buffer, filter cache, instruction register file, tagless hit instruction cache and revolver modified issue are designed for reducing instruction cache’s power dissipation. Researches on processors performance are also widely studied, such as temporal instruction fetch, proactive instruction fetch, shared history instruction fetch table. However, there are few researches making use of SPMD execution mode or focusing

on bank conflicts of shared instruction cache. Implementation complexity and hardware overhead are other disadvantages of previous works.

In this paper, we design an XOR-hash function to minimize access conflicts in the shared L1 instruction cache. In the design of XOR-hash function, under the premise of getting better randomization hash effects in the linear space of hash function, search cost and engineering implementation complexity are also taken into consideration. By adjusting transformation probabilities of the instruction cache queueing network, this XOR-based hash function can effectively reduce bank conflicts in the shared L1 instruction cache. Experimental results show that in a 4-core cluster with 32 KB instruction cache in all, the XOR hash-indexing shared instruction cache yields an 11% performance improvement compared to the private instruction cache and yields an 8% performance improvement compared to the set-interleaving shared instruction cache.

Our group has been working on the high-performance computing and processors design. We have developed several designs for instruction cache on home-grown heterogeneous many-core processor architecture. Such as instructions window buffer and level 0 instruction cache.

This work is supported by the National Key Technology Research and Development Program of China under Grant No. 2016YFB0200500. The target of this projects is to build validation system for key technology in exascale supercomputer by home-grown many-core processors and home-grown network processors.