一种片上众核结构共享 Cache 动态隐式隔离机制研究

宋风龙[1),2)] 刘志勇[1)] 范东睿[1)] 张军超[1)] 余 磊[1),2)]

[1)](中国科学院计算技术研究所计算机系统结构重点实验室 北京 100190)

[2)](中国科学院研究生院 北京 100039)

摘 要 访存带宽是限制众核处理器性能提升的关键,将片上最后一级 Cache 设计为所有处理器核共享是必要的.在共享 Cache 中隔离放置冲突的数据,是提高共享 Cache 性能的关键.文中提出了缓存块链接的硬件方法,用于隔离共享 Cache 中不同线程之间的数据.文中基于时钟精准的片上众核结构模拟器,使用 Splash2 程序组和生物信息学中的任务,对所提机制进行了评估.实验结果表明,与传统共享 Cache 相比,使用缓存块链接机制时,使得共享 Cache 的冲突性缺失率降低约 20%,而使得 IPC 平均提高了约 10%.

关键词 众核;共享 Cache;数据冲突;资源隔离;容量划分

中图法分类号 TP302 **DOI 号**: 10.3724/SP.J.1016.2009.01896

An Implicitly Dynamic Shared Cache Isolation in Many-Core Architecture

SONG Feng-Long[1),2)] LIU Zhi-Yong[1)] FAN Dong-Rui[1)] ZHANG Jun-Chao[1)] YU Lei[1),2)]

[1)](*Key Laboratory of Computer System and Architecture*, *Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190)

[2)](*Graduate University of Chinese Academy of Sciences*, *Beijing* 100039)

Abstract Memory bandwidth is critical to overall performance, especially for on-chip many-core architecture. It may be necessary to design a shared last level on-chip cache, to eliminate capacity wasted by multiple copies of one data block in private caches. However, when it comes to on-chip architecture, the conflict in shared cache becomes more serious than traditional single processor architecture. It is crucial to isolate conflicting data blocks in shared cache. This paper proposes a novel hardware approach, that is, block agglutinating, to isolate blocks of different threads in shared cache. Extensive analysis of the proposed scheme with Splash2 benchmarks and Bioinformatics workloads is performed using a cycle accurate many-core processor simulator. Experimental results show that when using block agglutinating, it makes an average reduction by about 20% in conflict miss rate of shared cache compared to the traditional shared cache, and it makes IPC improved by about 10%.

Keywords many-core architecture; shared cache; conflict; resource isolation; cache partition

1 引 言

由于传统超标量和深度流水单核处理器功耗和散热等问题,计算机体系结构发展的趋势是片上多核或众核处理器[1-4].这类处理器结构在同一个芯片内集成多个设计相对简单的处理器核,通过线程级并行得到性能提升的目的.随着半导体工艺的提升,

收稿日期:2009-07-15;最终修改稿收到日期:2009-08-20.本课题得到国家自然科学基金重点项目(60736012)、国家"九七三"重点基础研究发展规划项目基金(2005CB321600)、国家"八六三"高技术研究发展计划项目基金(2009AA01Z103)和北京市自然科学基金(4092044)资助.**宋风龙**,男,1980 年生,博士研究生,主要研究方向为高性能计算机体系结构、片上存储系统等. E-mail: songfenglong@ict.ac.cn. **刘志勇**,男,1946 年生,博士,研究员,博士生导师,主要研究领域为算法、计算机系统结构、并行处理、片上存储系统等. **范东睿**,男,1979 年生,博士,副研究员,主要研究方向为低功耗处理器设计. **张军超**,男,1976 年生,博士,研究方向包括计算机体系结构、先进编译技术. **余 磊**,男,1981 年生,博士研究生,主要研究方向为并行算法优化等.

芯片内可以集成的处理器核数目越来越多.但是,存储系统的访问速度并没有以相同的比例提升,因而处理器设计中遇到了文献[2]中提到的"存储墙"问题.在片上众核结构中,片上处理器核的处理能力和存储系统的访问速度之间的差距越来越大.因此为其设计高性能的片上存储系统至关重要,存储系统的两个基本要求是访问延迟更低以及片上有效容量更大.共享 Cache 比私有 Cache 的优势是可以有效利用容量,因而众核处理器设计的一个重要趋势是采用共享的最后一级片内 Cache[5].

然而,随着同时执行的处理器核数目的增加,共享 Cache 中的数据放置冲突也变得越来越严重,进而增加了存储系统的平均访问延迟,而影响了系统吞吐率和性能的进一步提升.为了避免共享 Cache 中的冲突,一种简单且有效的方法是隔离发生冲突的数据.其目的是减少在共享 Cache 中,同时执行的线程或程序之间发生数据污染的可能性,避免 Cache 访问抖动(thrash).本文有以下几点贡献:(1)通过区分 Cache 缺失的类别,说明众核处理器中不同线程在共享 Cache 级别发生线程交互的影响.冲突性缺失的进一步类别区分,有助于理解在共享 Cache 中避免数据放置冲突的必要性.(2)为了避免冲突,提出了一种称为缓存块链接的动态隔离机制.该机制基于硬件方式,在共享 Cache 中隔离不同线程的数据块,该机制的硬件实现开销可以忽略不计,并且不需要操作系统的支持.(3)详细分析了实验方法和结果.基于时钟周期精准的片上众核模拟器,使用 Splash2 测试程序组和生物信息学中的序列比对算法,对缓存块链接机制进行了评估.结果表明了该机制的有效性.

本文第 2 节分析本文的研究动机,说明本文关注此问题的原因;第 3 节介绍相关研究工作;第 4 节详细介绍所提出的共享 Cache 空间动态划分机制,即缓存块链接机制;第 5 节介绍本文的模拟平台和测试程序;第 6 节分析实验结果;第 7 节总结全文.

2　研究动机

文献[2]表明,众核结构中能够充分挖掘片上处理能力的编程模型为多线程模型.但是在支持多线程编程模型的片上众核结构中,共享 Cache 中的冲突会随着处理器核数目的增加而更为严重,如图 1 所示.众核结构的基本参数为:64 个处理器核;32KB,4 路组相联私有数据 Cache;16KB,2 路组相联私有指令 Cache;2MB,8 路组相联共享二级 Cache.将冲突性缺失进一步划分为线程内部(intra-thread)和线程之间(intra-thread)的冲突性缺失,如图 2 所示.结果表明,数据放置冲突占据大部分,尤其是线程之间的冲突性缺失;如果能够有效避免线程之间的数据放置冲突,会有效增大共享 Cache 的命中率.

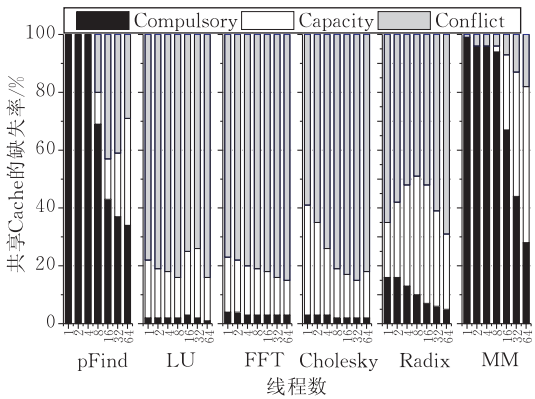

图 1　共享 Cache 缺失类别划分

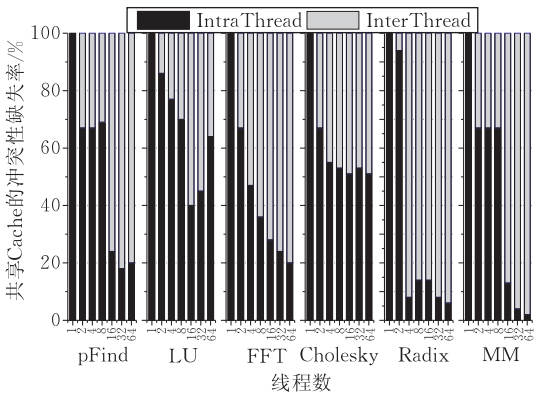

图 2　冲突性缺失类别划分

由于在程序执行期间,可能同时有多个处理器核访问共享 Cache,因而共享 Cache 成为片上网络中的一个热点(hot spot).若大部分对共享 Cache 的访问需要访问片外内存才能满足,片外内存的访问延迟相对而言非常大,因而将会对共享 Cache 的后续访问请求阻塞.文献[6]中将这种现象称为"树饱和"(tree saturation),指的是当路由到共享 Cache 的数据包被阻塞在某条路径上时,即使这些被阻塞的数据包和其后的数据包有不同的目的路由结点,也会进一步阻塞其后的数据包.由于树饱和会对后续路由的数据包有滚雪球效应,因而会显著降低片上网络的性能.因而增大片上存储的有效容量和降低片外缺失率非常重要,其目的是降低平均访存延迟,防止树饱和现象的出现.上述这些问题是本文关注片上众核结构共享 Cache 中数据隔离机制以避免数据放置冲突的主要原因.

3 相关研究

已有很多关于共享 Cache 冲突避免机制和容量划分机制的研究，如文献[7-13]等分别提出了静态或动态划分 Cache 容量的机制. 本节中，只介绍了和本文最相关的部分工作以及和本文机制的主要区别.

文献[5]中针对 CMP 提出了一种 Cache 结构，即基于处理器核的分散式共享 Cache(Shared Processor-based Split L2 Cache). 该机制基于统计信息(profile information)静态地将 Cache 以存储体为粒度分配给不同的应用程序. 其不足是需要在每次执行前，首先对所有的应用程序都统计(profile)执行状态. 而本文的机制则避免了这种统计行为. 文献[14]中提出了静态的 Cache 最优划分策略，其前提是需要动态统计每个应用程序可用 Cache 容量变化时相应 Cache 缺失的变化信息. 但是对于所有的应用程序，这种统计信息很难静态确定，因为与程序的输入集有关. 本文的目标是借助对缓存块的简单扩展，在运行时动态得到该信息. 文献[7]首次提出了共享 Cache 动态划分策略，该机制能够在同时执行的多个线程之间划分 Cache 容量. 该机制使用在 Cache 中命中的缓存块的位置预测每个应用程序对 Cache 的使用性. 其方法是在线程执行过程中，记录 Cache 缺失的类别，并根据操作系统的调度方式对 Cache 容量进行划分. 但是每个程序在共享 Cache 中命中的缓存块在信息栈中的位置受到其他线程的污染. 文献[15]提出了一种结合静态划分和动态划分的机制. Hsu 等在文献[16]中研究了不同的 Cache 划分策略，但是这些策略并不能根据程序的动态运行时状态进行划分. 文献[10]基于任务的使用性提出一种 Cache 动态隐式划分机制. 该机制需要为每个处理器核配备使用性监控电路 UMON，用于记录在每个处理器核上执行的程序对共享 Cache 的使用信息. 然后 Cache 划分算法根据 UMON 对所有处理器核统计的信息，对共享 Cache 以路为粒度进行划分. 文献[13]在多个线程之间划分共享 Cache 容量时的公平性. 文献[8]为共享 NUCA Cache 提出一种管理机制. 该机制基于片上网络互连的多存储体(bank)共享 Cache，并且以 Cache 存储体为粒度探讨共享度. 文献[12]提出一种共享 Cache 中的数据放置策略，该机制能够根据程序的执行行为做出反应. 本文所提缓存块链接机制与上述已有研究的区别之处在于，缓存块链接是一种以缓存块为粒度的、动态隐式的 Cache 划分机制，并且不需要操作系统的支持，因此对于片上众核结构而言非常简单且可扩展.

文献[17]提出一种自适应的组关联机制，和本文的缓存块链接类似. 二者的一个重要区别在于，文献[17]的组关联机制以共享 Cache 的组为粒度进行空间划分，而本文的机制是基于缓存块为粒度划分. 若 Cache 的相联度较大，组关联机制浪费的 Cache 容量较大. 因为在组关联机制中，一个组和某个特定的处理器核关联的一起，若线程自身的冲突度低于组相联度，则该组内空闲的缓存块会在程序占有该组的期间始终处于空闲状态，而浪费片上存储空间. 另外，组关联机制只能避免不同线程之间的冲突性缺失. 而本文的缓存块链接机制在增大片上有效容量的同时，既能避免不同线程之间的数据块放置性冲突，又能降低同一个线程数据块之间的放置性冲突. 另外，本文的缓存块链接机制不会对共享 Cache 空间进行严格显式的硬划分，每个处理器核的行为类似于从其它处理器核的共享 Cache 空间中“偷”部分容量. 因此缓存块链接能够更充分地使用共享 Cache 可用的总容量.

4 自适应缓存块链接机制

4.1 Cache 空间动态隔离机制

私有 Cache 的优势是不同线程之间不会发生数据放置冲突. 但是，由于每个处理器核只能使用整个二级 Cache 总容量的一小部分，会造成每个处理器核的容量性缺失(capacity miss)增大. 另外，私有 Cache 中同一个缓存块可能会存在多个副本，浪费片上存储容量的有效使用率，造成大量的片外缺失，而且维护二级 Cache 中的数据一致性也需要存储开销较大的目录等硬件结构. 因此众核结构的一个趋势是设计共享的片上最后一级 Cache. 但是，如图 1 所示，线程之间在共享 Cache 中的冲突非常严重. 增加共享 Cache 的相联度是降低冲突性缺失的一种简单方式，但是相联度越高表示功耗、设计复杂度和访问延迟都越高.

有许多研究针对片上多核结构提出了 Cache 容量划分策略，如第 3 节所述. 这些已有研究的主要不足是需要操作系统的支持，或者在片上众核结构中，其设计复杂度也是不能忽视的. 因此，本节提出了一种线程之间简单的 Cache 空间隔离机制，称为缓存块链接. 按缓存块为粒度对 Cache 空间进行动态隐式的划分，即某个处理器核将新读入的数据块放置

到 Cache 中某个缓存块的同时,表示该处理器核占用了该缓存块,在其占用期间其它处理器核无权将该缓存块替换出 Cache. 缓存块链接表示的是一种替换权,通过限制缓存块的替换关系,而在程序执行期间隐式地对 Cache 容量进行划分. 缓存块链接机制在缓存块放置和替换的同时,动态地在线程之间划分共享 Cache 空间、隔离不同线程的数据.

本文假设众核结构使用多线程执行模型,在某一个处理器核上执行的线程将会非抢占式地执行直至该线程释放处理器核,因此下文中不加区分地使用处理器核和线程两个词.

4.2　缓存块的自适应占用机制

如果线程按照先来先服务的策略占用缓存块,则可能会使得线程之间 Cache 空间划分不均匀. 因此需要设计为自适应的缓存块链接机制,即处理器核可以动态地释放部分被它占用的缓存块. 为了实现这一点,需要为每个缓存块扩展两个域:一个是信号计数器(signature counter),称为访问频率计数器(Access Frequency Counter, AFC);另一个是线程标识符,用于存放当前占用该缓存块的处理器核的标识符,如图 3 所示. 预先定义一个阈值,表示重新使用频率的最低下限值. 当某个 AFC 的值被减为比阈值低时,表示该缓存块的重新使用次数很少,其它处理器核可以抢占该缓存块用于缓存新读入的数据块. AFC 计数器值的更新是基于替换策略的,如图 4 所示. 每次重新访问并命中某个缓存块时,AFC 的值增 1. 每次在某个索引的组发生缺失时,该组内所有与当前发出请求的处理器核的 TID 不同的缓存块的 AFC 均减 1. 因此,AFC 计数器的值表示相应缓存块的重新使用频率. 当某个 AFC 的值低于阈值时,便将该缓存块的 TID 置为无效. 需要说明的一点是,缓存块的线程标识符 TID 置为无效,并不是将该缓存块的数据置为无效. 这种情况下,只是表明该缓存块可以被其它处理器核替换,但是已经缓存的数据块仍是有效的,并且直到被替换出 Cache 前,都可以被访问.

图 3　缓存块结构示意图

在本文模拟的基本平台中,为每个缓存块采用 4 位的 AFC 计数器,额外增加的硬件开销大约是共享 Cache 的 0.9%.

4.3　缓存块放置及替换策略

已有研究中有大部分是对共享 Cache 空间进行显式划分,另外也有部分基于使用程度对共享 Cache 进行隐式划分的机制,如第 3 节所述. 而缓存块链接机制则通过管理共享 Cache 的数据块放置和替换策略,隐式地在线程之间划分 Cache 空间和隔离缓存块.

当第一次访问某个数据而发生强制性缺失(compulsory miss)时,被请求的数据块从内存被载入到共享 Cache 中. 其内存地址确定了 Cache 中存放该数据块的索引位. 本节提出一种新的替换策略,称为 LRU-NoP,含义是在不同线程间没有数据污染前提下的类 LRU 替换策略. 放置新读入的数据块时,首先选择一个无效的缓存块. 若该组中没有无效的缓存块,则选择一个有无效 TID 的有效的缓存块,这表示该缓存块尽管有效,但是重新使用率较低,因此其拥有者线程释放了占用关系,因此这个缓存块可以被其它处理器核新读入的数据块替换. 否则,选择一个属于同一个线程的且使用频率最低的缓存块进行替换. 但是,若该线程占有的所有缓存块的使用频率均高于该组内其它线程的缓存块,则替换其它线程中使用频率最低的一个缓存块. 替换策略的流程图如图 4 所示.

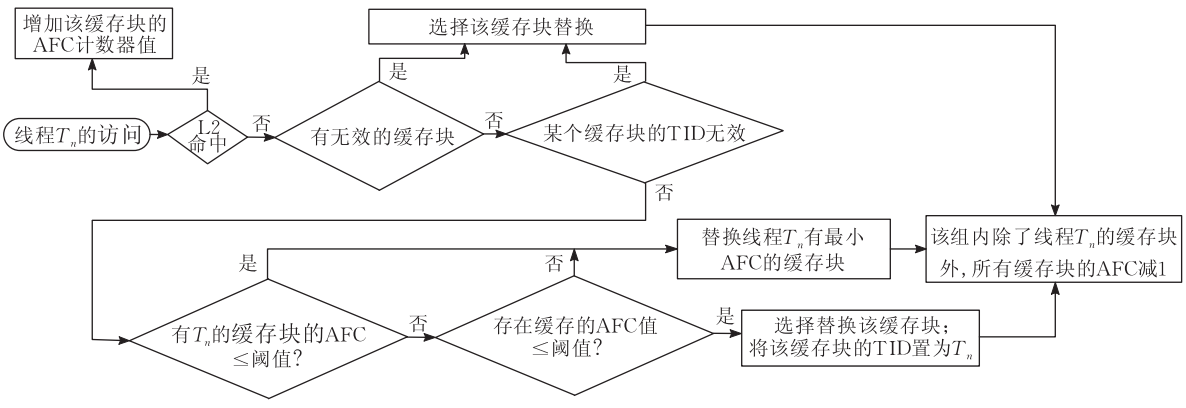

图 4　替换策略流程图

LRU-NoP 替换策略与传统替换策略(如 LRU)相比更为复杂.但是选择替换项的操作由组合逻辑实现,包括比较 TID 和 AFC 计数器的值,对时延的影响可以忽略不计;而对 AFC 计数器的更新并不在共享 Cache 回填一级私有 Cache 的关键路径上,因而对访存操作的延迟没有负面影响.在物理实现中,对于有 n 个处理器核的众核处理器而言,TID 域需要 $\log_2 n$ 位.在本文的基本模拟平台中,使用 64 个处理器核,TID 域的开销约为 1%.而 TID 域与 AFC 计数器二者的总硬件开销大约是共享 Cache 的 1.9%.

5 模拟方法

本节介绍本文的评估方法,包括评估平台和测试程序.

5.1 模拟平台

片上多核处理器的结构主要可以分为两类.第 1 类是"舞厅式"结构(dance-hall architecture),如 Sun UltraSparc T1/T2[18] 和 Cyclops[1];第 2 类是分片式结构(tiled architecture),如 MIT 的 RAW 处理器[19],即 Tilera 公司的 Tile64 处理器.前者的主要优势是共享 Cache 中的数据可以以较低的访问延迟索引.但是,其不足是不易于扩展,因为随着芯片规模的增大,片上集中式互连结构的连线延迟也相对增大.而分片式结构则由于采用片上网络而易于扩展.但是其主要不足是内存控制器的设计复杂度高,并且数据索引定位和放置等复杂性较高.

为了结合上述两种结构的优点,我们提出一种混和式结构,即使用片上互连网络和相对集中的共享存储,如图 5 所示.该模拟器称为 Godson-T,是时钟周期级准确的同构众核结构模拟器.其中,芯片内集成了 64 个顺序双发射的处理器核,每个处理器核基于 MIPS II 结构.每个处理器核有独立的私有一级数据和指令Cache以及程序员可编程控制的Scratch-Pad Memory.片内集成的有共享的静态 NUCA 结构 Cache、同步管理器(synchronization manager)和内存控制器,并使用 8×8 的 2-D Mesh 作为片上互连网络.Godson-T 上执行的应用程序通过一个轻量级的运行时系统(runtime)管理,负责线程的创建、分配和收集等工作.

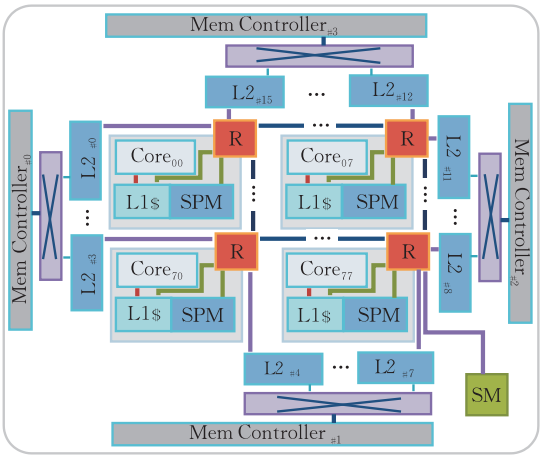

图 5 众核结构模拟器示意图

Godson-T 模拟器的二级共享 Cache 设计为 16 个存储体,且分布在芯片的四周,由所有处理器核共享.每个存储体连接到 Router 和一个 4 端口的交叉开关.其中,Router 将每个存储体连接到片上网络中,并且某个处理器核访问不同 Cache 存储体的时间是不同的,取决于二者之间的物理距离;而交叉开关则将 Cache 存储体与内存连接在一起,负责二者之间的数据交换.二级 Cache 设计为非阻塞结构,即每个存储体均可以同时处理若干个 outstanding cache misses.片外内存的最大存储带宽是 32GB/s.

我们提出的这种混和式众核结构主要有 3 个优点,即可扩展性好、访问延迟低和片上有效容量大.该结构的有效性在文献[20]中进行了评估.本文评估中的结构参数如表 1 所示.

表 1 模拟结构参数

模块	结构配置参数
处理器核	8 级流水,顺序双发射,2 ALU&FPU,1 DIV/MUL,非抢占单线程
一级数据 Cache	私有,32KB,32B 缓存块,4 路组相联,命中延迟 1cycle
SPM	通过一级数据 Cache 配置,访问延迟 1 cycle
一级指令存储	私有,16KB,32B 缓存块,2 路组相联,命中延迟 1cycle
二级 Cache	共享,总容量 2MB,16 个 Bank (128KB/bank),64B 缓存块,8 路组相联,16×4 项请求队列,支持非阻塞式 out-standing miss
片上互连网络	8×8 2-D mesh,静态 X-Y 路由策略,2cycles/跳
路由器	2 级流水,2 个虚通道,128 位数据带宽
内存控制器	4 个,512 位数据带宽,读延迟 52cycles,写延迟 32 cycles

5.2　测试程序简介

本文选择了 5 个科学计算的内核算法,即矩阵乘(MM)和 Splash2 测试程序集[21]中的 FFT、LU 分解、Radix 和 Cholesky 等,以及生物信息学中序列比对算法 pFind[22]. 其中 pFind 是串连质谱中自动识别肽和蛋白质的算法,该算法是存储访问密集型的程序,可以充分评估存储系统. 另外,其它 5 个算法是高性能科学计算中常用的算法.

本文测试的程序规模如表 2 所示. 所有测试程序都执行结束,并且在不预热 Cache 的前提下,在执行期间统计性能数据.

表 2　程序规模

测试程序	程序规模	优化选项	指令数/10^6
pFind	32768 个肽序列	−O3	547.49
FFT	1048576 个双精度复数	−O3	101.8
LU	512×512 矩阵, 16×16 元素块	−O3	618.2
Cholesky	输入: tk15.O	−O3	1048.29
Radix	1M 键值,1024 点排序	−O3	318.31
MM	256×256 双精度浮点矩阵	−O3	10.9

6　实验结果分析

由于大部分已有的 Cache 划分机制需要操作系统支持,并且物理实现非常复杂. 或者以处理器核为中心构造共享 Cache 划分机制,这种方案在片上处理器核数目较少的多核处理器中实现开销相对较小,但是在众核结构中,其硬件实现开销非常大. 因此本节并不与这些已有工作的性能进行比较.

6.1　共享 Cache 缺失率

因为本文的机制中并未考虑预取,因此该机制不会影响 Compulsory Miss 的数目. 缓存块链接机制对共享 Cache 缺失率的影响如图 6 所示. 缺失率相对于传统共享 Cache 进行归一化表示. 结果表明,本文所提机制可以改善除了 Cholesky 和矩阵乘 MM 两个程序外其它程序的缺失率. 对 Choloesky 缺失率改善不佳的主要原因是该程序有较多互相冲突的缓存块被映射到少数不同的索引位,使得这些缓存块总是频繁的彼此替换. 但是使用随机替换算法时,该问题不会出现. 对于矩阵乘程序而言,其冲突性缺失所占的比例相对较小. 原因是本文测试采用的矩阵规模较小,其工作集较小,导致其大部分缺失为强制性缺失,因此改善也不明显. 平均而言,缓存块链接机制可以使得共享 Cache 的缺失率降低 3%.

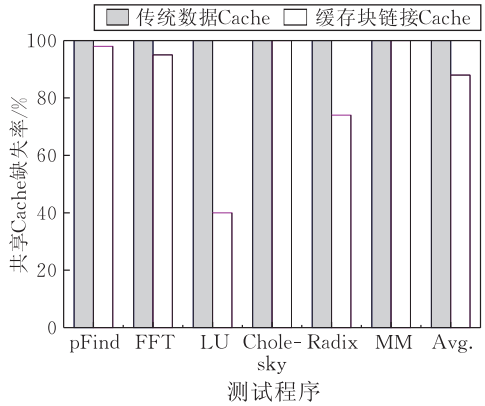

图 6　共享 Cache 缺失率

6.2　共享 Cache 冲突性缺失率

缓存块链接机制主要用于避免不同线程在共享 Cache 中的冲突,对共享 Cache 冲突性缺失降低的情况如图 7 所示. 结果相对于传统共享 Cache 进行归一化处理. 尽管图 6 表明本文机制对矩阵乘的总体缺失率影响不大,但是可以看出其对所有测试程序共享 Cache 的冲突性缺失的降低非常大. 尤其对于矩阵乘而言几乎避免了所有的冲突性缺失. 平均而言,所提机制使得共享 Cache 的冲突性缺失率降低了 20%,平均占冲突性缺失的 40%.

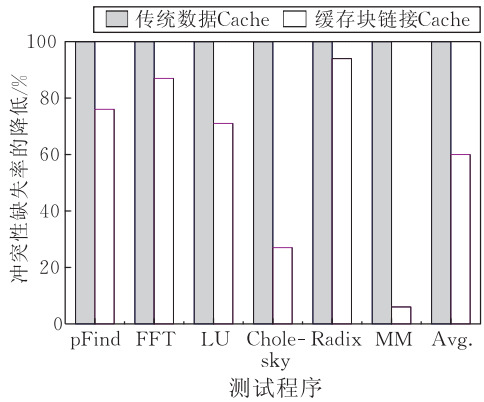

图 7　冲突性缺失的减少

6.3　共享 Cache 的平均读延迟

如图 8 所示,缓存块链接机制显著降低共享 Cache 的平均读延迟. 结果与传统共享 Cache 做归一化处理. 读延迟指的是自共享 Cache 接收到读请

求，至共享 Cache 发出回填数据到 Router 为止所用的时钟周期. 在这个过程中，可能会发生由于共享 Cache 缺失而对片外内存的读取请求. 结果表明，所提机制能显著降低共享 Cache 的平均读延迟，尤其对于 LU 和 Radix 更为明显. 但是与上述相同的原因，对 Cholesky 和矩阵乘 MM 的读延迟改善并不明显. 平均而言，所提机制使得共享 Cache 的平均读延迟降低了 17%.

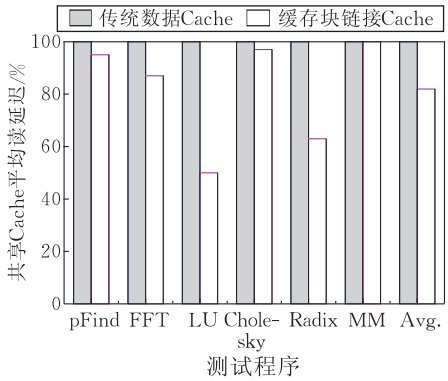

图 8 共享 Cache 读延迟

6.4 私有数据 Cache 的平均读延迟

一级私有 Cache 的读延迟对处理器总体性能的影响非常大. 平均读延迟包括一级 Cache 的命中访问延迟和缺失处理开销. 如图 9 是与传统共享 Cache 归一化后的结果. 随着处理器核数目的增加，一级私有 Cache 和下一级共享 Cache 之间的通信开销增大，原因是片上互连网络的通信压力增大. 结果表明，所提机制能显著降低一级数据 Cache 的平均读延迟，尤其对于 FFT 和 Radix 更为明显. 平均而言，所提机制使得私有数据 Cache 的平均读延迟降低了 1.5 个时钟周期，大约降低了 11%.

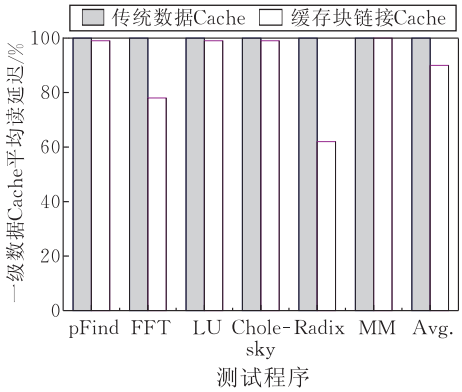

图 9 L1 平均读延迟

6.5 私有数据 Cache 的平均缺失开销

一级私有数据 Cache 处理缺失的平均开销所占总执行时间的比例，反映了缓存块链接机制对整体性能的影响. 私有 Cache 处理缺失的开销指的是，自私有 Cache 发出缺失请求直至下一级共享 Cache 将回填数据完全发回来这期间的时钟周期. 私有 Cache 处理缺失的平均开销所占总执行时间的比例如图 10 所示，其结果相对于传统共享 Cache 进行归一化处理. 结果表明所提机制对除矩阵乘 MM 以外的几个测试程序都显著减少了该比例. 平均而言，所提机制将私有 Cache 处理缺失的平均开销所占总执行时间的比例降低了约 9%.

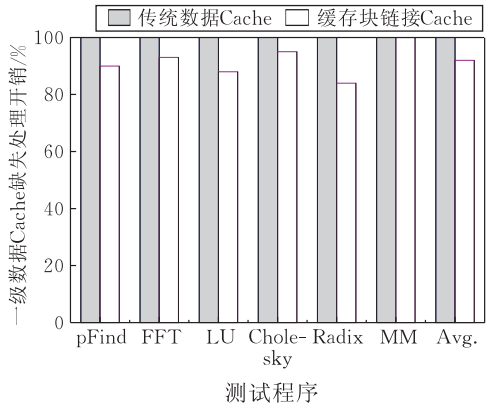

图 10 L1 缺失开销比例

6.6 IPC

每个时钟周期执行的指令数(Instructions per Cycle，IPC)是另一个反映整体性能的重要度量标准. 所提机制与传统共享 Cache 的 IPC 比较如图 11 所示，结果相对于传统共享 Cache 做归一化处理. 结果表明缓存块链接机制能改善除了 LU 分解外其它测试程序的 IPC. 对 LU 分解效果不明显的原因是 LU 分解中有大量的同步操作，导致大量在片上互连网络中的消息包，从而使得 LU 的 IPC 有少许降低. 但是平均而言，所提机制能使 IPC 提高约 10%.

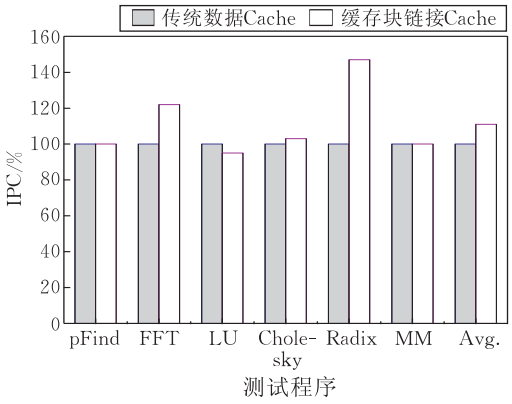

图 11 IPC

总体而言，对于大多数所评估的程序，所提出的缓存块链接机制是一种简单但是有效的共享 Cache

空间隔离机制. 对于矩阵乘 MM,实验结果表面看来所提机制的性能和传统共享 Cache 相当;原因是本文所评估的矩阵规模较小,其工作集较小,因而产生的冲突性缺失相对较少,因此整体性能改善不明显. 另外,尽管本文只针对片上众核结构中的静态 NUCA Cache 结构进行评估,但是该机制可以轻松地扩展到其它 Cache 结构.

7　结论和下一步工作

对于片上众核结构而言,片上最后一级 Cache 由所有处理器核共享是非常必要的,以便于有更大的有效容量和更少的片外缺失率. 但是,对于支持多线程编程模型的众核结构而言,线程之间的冲突会变得非常严重. Cache 空间隔离划分是解决线程在共享 Cache 中数据污染的简单有效机制. 针对于此,本文主要工作如下:首先,在众核结构共享 Cache 中,区分了 Cache 缺失的类别,表述了多个线程之间在共享 Cache 中的交互作用;其次,为了避免多个线程之间的冲突,本文提出了一种隐式动态的划分共享 Cache 空间的机制,即缓存块链接机制,用于隔离不同线程在共享 Cache 中的数据. 该机制硬件实现开销可以忽略不计,并且不需要操作系统的支持,更有利于片上众核系统的可扩展性要求;最后,详细介绍了评估方法和实验结果. 结果表明了本文机制的有效性.

尽管所提出的机制能显著减少共享 Cache 的冲突性缺失率,但是随着处理器核数目的增加,线程之间的冲突会变的越来越严重. 因此,下一步将采用专用硬件结构扩展 Cache 隔离机制,降低由于在共享 Cache 中的冲突而造成的副作用,进一步降低平均访存延迟.

参　考　文　献

[1] Almasi G, Cascaval C et al. Dissecting cyclops: A detailed analysis of a multithreaded architecture. ACM SIGARCH Computer Architecture News, 2003, 31(1): 26-38

[2] Asanovic K et al. The landscape of parallel computing research: A view from berkeley. UC Berkeley, Technical Report: No. UCB/EECS-2006-183, December 18, 2006

[3] Olukotun K, Nayfeh B A et al. The case for a single-chip multiprocessor. ACM SIGPLAN Notices, 1996, 21(9): 2-11

[4] Seiler L, Carmean D et al. Larrabee: A many-core X86 architecture for visual computing. ACM Transactions on Graphics, 2008, 27(3): 1-15

[5] Lin C, Sivasubramaniam A et al. Organizing the last line of defense before hitting the memory wall for CMPs//Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA'004). Washington, DC, USA: IEEE Computer Society, 2004: 176-185

[6] Pfister G F, Norton V A. "Hot-spot" contention and combining in multistage interconnection networks. IEEE Transactions on Computers, 1985, C-34(10): 943-948

[7] Suh G E, Rudolph L, Devadas S. Dynamic partitioning of shared cache memory. The Journal of Supercomputing, 2004, 28(1): 7-26

[8] Huh J, Kim C et al. A NUCA substrate for flexible CMP cache sharing//Proceedings of the 19th International Conference on Supercomputing (ICS'05). Boston, MA, USA, 2005: 31-40

[9] Chang J, Sohi G S. Cooperative cache partitioning for chip multiprocessors//Proceedings of the 21st ACM International Conference on Supercomputing. Seattle, 2007: 242-252

[10] Qureshi M K, Patt Yale 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. Washington, DC, USA: IEEE Computer Society, 2006: 423-432

[11] Jouppi N P. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers//Proceedings of the ISCA'90. ACM SIGARCH Computer Architecture News, 1990, 18(3a): 364-373

[12] Hardavellas N, Ferdman M et al. Reactive NUCA: Near-optimal block placement and replication in distributed caches//Proceedings of the ISCA 2009. Austin, TX, USA, 2009: 184-195

[13] Kim S, Chandra D, Solihin Y. Fair cache sharing and partitioning in a chip multiprocessor architecture//Proceedings of the 13th International Conference on Parallel Architecture and Compilation Techniques (PACT'04). Washington, DC, USA: IEEE Computer Society, 2004: 111-122

[14] Stone H S et al. Optimal partitioning of cache memory. IEEE Transactions on Computers, 1992, 41(9): 1054-1068

[15] Iyer R. CQoS: A framework for enabling QoS in shared caches of CMP platforms//Proceedings of the ICS-18. Malo, France, 2004: 257-266

[16] Hsu L R et al. Communist, utilitarian, and capitalist cache policies on CMPs: Caches as a shared resource//Proceedings of the PACT-15. Seattle, Washington, USA, 2006: 13-22

[17] Srikantaiah S, Kandemir M, Irwin M J. Adaptive set pinning: Managing shared caches in chip multiprocessors//ACM SIGOPS Operating Systems Review (ASPLOS'08). New York, NY, USA: ACM, 2008, 42(2): 135-144

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

[19] Taylor M B et al. Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ILP and streams//Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA-31). Munchen, Germany, 2004: 2-13

[20] Wang X et al. A quantitative study of the on-chip network and memory hierarchy design for many-core processor//Proceedings of 14th IEEE International Conference on Parallel and Distributed Systems. Washington, DC, USA: IEEE Computer Society, 2008: 689-696

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

[22] Fu Y, Yang Q et al. Exploiting the kernel trick to correlate fragment ions for peptide identification via tandem mass spectrometry. Bioinformatics, 2004, 20(12): 1948-1954

SONG Feng-Long, born in 1980, Ph. D. candidate. His research interests include high performance computer architecture, and on-chip memory systems.

LIU Zhi-Yong, born in 1946, Ph. D., professor, Ph. D. supervisor. His research interests include algorithm, parallel processing and memory systems.

FAN Dong-Rui, born in 1979, Ph. D., associate professor. His research interests include low-power design and processor micro-architecture.

ZHANG Jun-Chao, born in 1976, Ph. D.. His research interests include computer architecture, compiler design and runtime systems.

YU Lei, born in 1981, Ph. D. candidate. His research interests focus on parallel algorithm.

Background

The many-core architecture is becoming a promising computing platform for transistors integrated into one single chip increased greatly due to the advancement of semi-conductor technology. Many-core architecture is different from multi-core architecture from nature of programming and design consideration, and there are many research groups coming from different institutes and universities are dedicated to it, such as, MIT presented tiled CMP/many-core architecture named RAW, Cyclops-64 of IBM and in 2007, Intel announced their 80-core TeraFlops prototype chip in ISSCC conference.

In present, topics on on-chip memory systems of many-core architecture are mainly research papers in most important conference and journal, and almost all of these researches are aimed to optimize on-chip cache based on shared or private baseline. And the research point is to reduce access latency of memory systems and control wire delay properly, for wire delay has become significant in many-core chip.

As far as our knowledge, the last level cache of almost all of emerging many-core architectures from literatures and prototypes is designed to be shared by all processors on-chip. Now it is the question, that is, when the number of on-chip threads running concurrently is increasing, the conflict between these threads at the shared cache is worse, and there are few researches on it. The main disadvantage of previous works about cache partition is the implementation complexity and hardware overhead. In addition, they need support of operation system. So this paper presents an implicitly dynamic cache partition scheme, which can be implemented with negligible hardware cost and without support of operation system, and evaluates the proposed scheme via cycle accurate many-core architecture simulator Godson-T.

This work is under the support of the National Natural Science Foundation of China (grant No. 60736012), National Basic Research Program (973 Program) of China (grant No. 2005CB321600), National High Technology Research and Development Program (863 Program) of China (grant No. 2009AA01Z103), and Beijing Natural Science Foundation (grant No. 4092044).