计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20150140
52(6):1242-1253,2015

多核处理器目录缓存结构设计

王恩东　唐士斌　陈继承　王洪伟　倪　璠　赵雅倩
(高效能服务器和存储技术国家重点实验室(浪潮集团有限公司)　北京　100085)
(wangend@inspur.com)

Directory Cache Design for Multi-Core Processor

Wang Endong, Tang Shibin, Chen Jicheng, Wang Hongwei, Ni Fan, and Zhao Yaqian
(*State Key Laboratory of High-End Server & Storage Technology* (*Inspur Group Company Limited*), *Beijing* 100085)

Abstract　With the development of Internet of things, cloud computing and Internet public opinion analysis, big data applications are growing into the critical workloads in current data center. Directory cache is used to guarantee cache coherence in chip multi-processor, which is massively deployed in data centers. Previous researches proposed all kinds of innovation to improve the utilization of directory cache capacity and scalability, making it more suitable for high-performance computing. Big data workloads are timing sensitive, which is not satisfied by previous works. To meet the requirement of big data workloads, master-salve directory is a novel directory cache design, which can optimize the path of memory instruction. In the novel directory cache design, master directory picks up private data accesses and provides services for them to reduce miss-latency, and slave directory provides cache coherence for shared memory space to improve the utilization of cache capacity and the scalability of chip multi-processor. Our experiment benchmark is CloudSuite-v1.0, running on Simics + GEMS simulator. Compared with sparse directory with 2 × capacity, the experimental results show that master-slave directory can reduce hardware overhead by 24.39%, and reduce miss-latency by 28.45%, and improve *IPC* by 3.5%. Compared with in-cache directory, the results show that master-slave directory sacrifices 5.14% miss-latency and 1.1% *IPC*, but reduces hardware overhead by 42.59%.

Key words　big data; multi-core processor; cache coherence; directory cache; sparse directory

摘　要　随着物联网、云计算与网络舆情分析等应用的快速发展,大数据处理的应用已经成为数据中心的核心负载.数据中心服务器普遍采用多核处理器,而目录缓存作为多核处理器结构中维护缓存一致性的关键部件,对其结构研究(如稀疏目录)更多地关注于目录缓存的容量与可扩展性,更适合处理高性能计算等计算密集型应用.然而,当多核处理器执行延迟敏感的大数据应用程序时,目录缓存的高访存延迟严重制约了数据中心的服务质量.针对该问题,新型主从目录缓存结构优化了数据访问过程中的一致性协议通路,其中主目录区分共享与私有数据,管理私有数据的访存操作,降低私有数据的访存延迟,提高了从目录的容量利用率;从目录维护共享数据的缓存一致性,采用有限位标签结构,提高了从目录的存储效率.实验在Simics+GEMS模拟平台上对大数据程序测试集Cloudsuite-v1.0进行评估.结果表明在以大数据应用程序为主的运行环境下,与2倍容量的稀疏目录相比,主从目录缓存结构降低了24.39%的硬件开销,降低了28.45%的缓存缺失延时,提升了3.5%的处理器*IPC*;与缓存内目录相比,主从目录结构虽然损失了5.14%的缓存缺失延时与1.1%的处理器*IPC*,但是降低了42.59%的硬件开销.

收稿日期:2015-02-13;**修回日期**:2015-04-21
基金项目:国家"八六三"高技术研究发展计划基金项目(2013AA011701)
通信作者:唐士斌(tangshb@inspur.com)

关键词 大数据;多核处理器;缓存一致性;目录缓存;稀疏目录

中图法分类号 TP303

随着物联网、云计算、网络舆情分析等应用的快速发展,数据中心所承载的数据量呈爆炸式增长,导致处理大规模数据的大数据应用程序成为数据中心的核心负载.在当前的数据中心体系结构下,多核处理器作为数据处理的核心部件,其处理大数据应用程序的能力直接决定了数据中心的服务质量.然而,文献[1]认为多核处理器的存储层次、前端流水线设计以及处理器核的执行部件设计更适合传统的计算密集型应用,不能有效地利用片内资源来处理大数据应用.此外,通过分析典型大数据应用程序发现传统一致性目录缓存结构存在数据访问延迟大、存储资源利用率低等问题.

在多核结构中,侦听与目录是缓存一致性协议的2种重要的实现方式[2].由于侦听协议的可扩展性差,具有更高可扩展性的目录协议得到了广泛的使用.目录缓存是片上多核处理器实现目录协议的核心部件,其设计与实现主要有3种方式:复制标签目录[3-4]、缓存内目录[5]、稀疏目录(sparse directory)[6-7].1)复制标签目录结构,复制了片内私有缓存的标签域,来维护缓存一致性;其优点是硬件开销小、无目录替换;其缺点是查找功耗高、可扩展性差.2)缓存内目录利用部分末级缓存(last-level-cache, LLC)的硬件逻辑,与LLC紧耦合设计实现;其优点是存储利用率高且目录缓存的替换率低;其缺点是容量利用率低,造成大量的空间浪费且难以扩展.3)稀疏目录采用独立的缓存结构来记录数据副本在处理器核之间的分布情况;其优点是可扩展性好、查找功耗低、容量利用率高;但是由于稀疏目录的组相联度低,容易产生缓存组内的冲突替换,导致L1缓存的抖动问题.针对稀疏目录的冲突替换问题,学术界进行了大量的研究,如Cuckoo Directory[6],Deactivating Directory[7]等.这些研究成果在降低稀疏目录缓存的冲突替换方面取得了良好的实验效果.

然而,稀疏目录缓存结构虽然降低了目录缓存查找功耗、提高了缓存容量利用率,但是其增加了处理器访问数据操作的延迟.受到商业处理器存储一致性模型的约束,维护私有缓存的数据一致是保证多线程程序正确执行的关键.为了保证私有缓存的数据一致性,在实现目录缓存的处理器中,所有访存操作在访问共享存储区域(如末级共享缓存、共享内存)之前,必须首先访问目录缓存结构,执行一系列的一致性操作[2].该过程增加了处理器访问数据的关键路径长度,导致访存操作延迟增加,降低了多核处理器执行访存密集型应用时的性能.而目前的大数据应用,不仅是访存密集型应用,而且是延迟敏感型,如视频流媒体、网页搜索、数据服务等.而目录缓存结构增加了数据访问延迟,导致片上多核处理器处理数据的延迟增加,严重影响执行上述应用的数据中心服务质量.

通过分析典型的大数据应用程序,发现大数据应用程序以对私有数据的访问为主,而在共享存储编程模型中,私有数据没有维护缓存一致性的必要性.基于此特征,针对现有设计中存在的问题,本文系统地考虑了处理器的数据访问延迟、目录缓存的硬件开销以及目录缓存的容量利用率等方面因素,提出了主从目录缓存结构.在基于主从目录缓存结构的处理器中,处理器访问的数据被分成了共享与私有2类.对于共享数据执行正常的一致性维护流程;而对于私有数据则直接访问共享存储区域,避免了对目录缓存结构的访问,降低了访问私有数据的延迟.对大数据应用程序而言,访问私有数据的操作占所有访问内存操作的比重较大,降低了访问私有数据的延迟,可以有效提升系统的整体性能.

在主从目录缓存结构设计中,主目录负责共享与私有数据的区分,并为私有数据访问提供数据服务.主目录与末级缓存紧耦合设计,在末级缓存的缓存行中增加2个域:私有(或共享)数据标记与私有数据副本指针,用来记录缓存行是否处于私有状态以及私有数据副本的位置.从目录与主目录共同维护共享数据的一致性,从目录采用有限位标签的结构设计,降低稀疏目录设计的硬件开销.当数据处于共享时,根据系统配置的不同,从目录或者基于有限位标签或者基于主目录的私有数据副本指针,判断共享信息是否被从目录缓存.

本文的主要贡献有3点:

1) 提出了主从目录缓存结构,降低了私有数据的访存延迟,提高了从目录的容量利用率.

2) 提出了有限位标签的从目录结构,降低了从目录的硬件设计开销,提高目录缓存的存储效率.

3) 提出了基于时间窗口的共享与私有数据划分方法,解决了严格区分共享与私有数据难以在硬件层面实现的问题.

1 研究背景与动机

高效地维护缓存一致性是保证多核处理器性能与可扩展性的关键. 与基于侦听的缓存一致性协议实现相比,基于目录的协议实现具有效率高、可扩展性高以及网络开销低等优势,因此被商业处理器厂商广泛采用,如 Oracle[3-4] 与 Intel[5] 等. 然而在实际的系统中,目录对存储空间的需求很大,因此往往被保存在片外存储,导致目录访问存在延迟高、带宽低等问题. 目录缓存是目录在处理器内部的缓存空间,可以有效降低处理器访问片外目录的次数,其执行效率是影响目录协议性能的重要因素. 稀疏目录是目录缓存结构的一种重要的实现方式,本节重点介绍稀疏目录缓存结构设计以及目录缓存设计在大数据环境下面临的新挑战.

1.1 基础平台介绍

本文以图 1 所示的处理器结构作为基础平台(末级缓存与私有缓存之间是全包含关系),来介绍稀疏目录缓存的结构设计以及本文提出的主从目录结构的核心思想与实现方式. 根据文献[1,8]所述,传统多核处理器的体系结构设计并不适合数据中心的大数据应用,如深的存储层次、复杂的处理器核以及大容量的末级缓存等. 而如图 1 所示的两级存储层次、精简的小核结构以及适当容量的末级缓存,更适合大数据处理,因此将该结构作为本研究的基础研究平台.

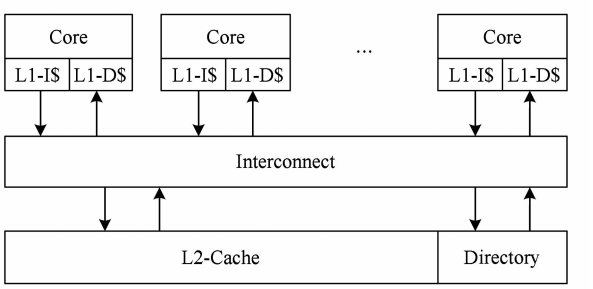


Fig. 1 Memory hierarchy of baseline system.

图 1 基础平台的存储层次示意图

1.2 稀疏目录结构与数据通路设计

1.2.1 稀疏目录结构

文献[9]最早提出了稀疏目录结构. 稀疏目录是具有独立参数配置空间的片上存储区域,其中可配置的参数有组相联度、索引位长度等. 稀疏目录采用多路组相联结构,每个目录项的缓存行包含 3 个域:标签、状态位与共享副本向量. 其中,标签域用来检查缓存是否命中;状态域用来记录数据的一致性状态;共享副本向量用来记录缓存数据副本在处理器核之间的分布情况. 由于多核结构中私有缓存的容量有限,因此存在仅有冷缺失的稀疏目录结构设计. 例如,片上多核处理器包含 N 个处理器核,每个处理器核的私有缓存采用 W_{L1} 路组相联结构,缓存行数量为 B. 理想稀疏目录的缓存行总数应为 $N\times B$,采用 $N\times W_{L1}$ 路组相联. 然而上述结构设计组相联度高、查找功耗高,因此实际的稀疏目录往往采用较低的组相联度、更大的缓存容量,如 $2\times W_{L1}$ 路组相联结构,$2\times N\times B$ 个缓存行.

1.2.2 稀疏目录的数据访问通路

稀疏目录结构增加了处理器的数据访问延迟,如图 2 所示. 该示例展示了当片上存在有效数据副本时基于稀疏目录结构的读操作数据访问流程.

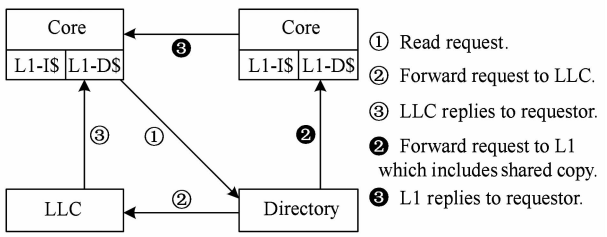


Fig. 2 Data path of read operation in sparse directory.

图 2 传统稀疏目录结构的读操作数据通路

在图 2 的示例中,如果末级缓存(LLC)存在有效数据副本,则对应的数据访问过程为①→②→③. 其中,步骤①,处理器核发起访问请求;步骤②,目录缓存转发请求到末级缓存;步骤③,末级缓存回复应答消息. 如果末级缓存中不存在有效副本,对应的数据访问过程为①→❷→❸. 其中,步骤①,处理器核发起访问请求;步骤❷,目录缓存转发请求到持有副本的 L1 缓存;步骤❸,L1 缓存回复应答消息.

在上述执行过程中,由于稀疏目录的存在,导致数据访问流程分 3 个阶段. 然而对于私有数据而言,其没有维护一致性的需求. 这种 3 段式的访问方式,增加了数据访问延迟,进而损害了处理器执行访存密集型应用时的整体性能.

1.3 大数据应用带来的设计挑战

文献[10]指出维护缓存一致性在未来的处理器设计中仍将占有重要的位置,并且提高维护缓存一致性的效率是目前多核处理器结构设计的重要问题. 以稀疏目录为代表的目录缓存结构是商业处理器中维护缓存一致性的重要模块,当面对数据中心的大数据应用时,其结构设计面临 2 个重要的挑战:

1) 稀疏目录结构不能满足大数据应用对数据访问延迟的要求.与传统计算密集型的应用不同,大数据应用更多是具有时间约束的访存密集型应用.数据访问的延迟问题已经成为商业服务器设计的关键问题之一.以 Facebook(脸谱)公司的页面访问过程为例,一次访问将导致大量连续的存储资源的访问,而数据访问的延迟导致 Facebook 必须将在关键路径上的服务器访问数量限制在 150 个以内[11].而在多核处理器设计中,一方面,稀疏目录增加了访存关键路径长度,为了避免并发访问引入的高功耗问题,末级缓存内部以及末级缓存与稀疏目录之间都是采用串行访问,导致数据访问延迟增大;另一方面,稀疏目录容易造成 L1 缓存的抖动,进一步增大数据访问延迟.

2) 稀疏目录的存储开销不能满足大数据应用对系统可扩展性的需求.系统的扩展能力不仅是数据中心体系结构的一个重要特征,同样也是数据中心处理器的一个关键属性[8].然而在保证维护一致性协议执行效率的前提下,稀疏目录的存储开销以处理器核数平方的速度增长:一方面,共享副本向量长度随着核数线性增加;另一方面,要保证一致性效率,稀疏目录项数随着核数线性增加.因此,降低稀疏目录的硬件开销是提高稀疏目录可扩展性的重要方面.

针对处理大数据应用时稀疏目录结构存在的上述 2 个问题,结合大数据应用程序以私有数据访问为主的特点[1],本文提出了面向大数据应用的主从目录缓存结构,来降低目录协议中的对私有数据访问的延迟,提高目录缓存的容量利用率和存储效率.

2 主从目录缓存结构

本节重点介绍主从目录结构的核心思想与实现方法.首先,介绍主从目录的数据访问通路,通过优化处理器的数据访问通路,解释了主从目录如何降低访问私有数据的延迟;然后,描述主目录组织结构,并针对硬件层面难以区分共享与私有数据,提出了基于时间窗口的共享数据识别方法;最后,介绍基于有限位标签的新型从目录结构,降低目录缓存设计的硬件开销.

2.1 主从目录的数据访问通路

针对稀疏目录缓存结构增加了处理器访问数据延迟的问题,基于大数据应用程序以私有数据访问为主的特点,提出了主从目录缓存结构,分别为私有数据与共享数据配置不同的数据访问通路,并缩短私有数据访问的关键路径长度以降低平均数据访问延迟.图 3 与图 4 分别展示了主从目录结构对私有数据以及共享数据的访问通路.在示例中,主目录(master directory)采用缓存内目录结构,用来记录私有数据副本在私有缓存的分布情况;从目录(slave directory)采用稀疏目录结构,用来记录共享数据副本在私有缓存的分布情况.

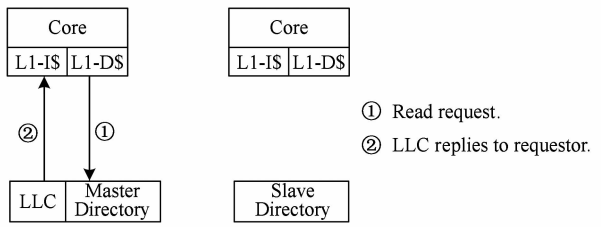


Fig. 3 Data path of accessing private memory space.

图 3 读取私有数据的数据访问通路

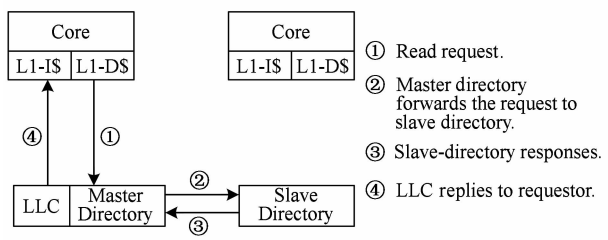


Fig. 4 Data path of accessing shared memory space.

图 4 读取共享数据的数据访问通路

读取私有数据的数据访问过程:①→②,如图 3 所示.其中,步骤①,处理器核发起访问请求;步骤②,主目录识别该请求访问私有数据,LLC 直接提供数据服务.在该过程中,与传统稀疏目录结构的数据访问过程相比,主从目录结构可以避免对从目录的访问,减小了关键路径长度,降低了访问私有数据的延迟.

读取共享数据的数据访问过程:①→②→③→④,如图 4 所示.其中,步骤①,处理器核发起访问请求;步骤②,主目录识别该请求访问共享数据,转发维护一致性请求到从目录;步骤③,从目录更新共享副本向量并回复应答;步骤④,持有数据副本的末级缓存应答数据请求.在该过程中,与传统的稀疏目录结构相比,增加了与从目录的交互,导致访问共享数据的延迟增加.

尽管主从目录的数据通路损害了访问共享数据的延迟,但是已有的研究工作[1]以及本文的实验结果均显示大数据应用程序以私有数据访问为主,因此降低对私有数据的平均访问延迟可以降低整个系统的数据访问延迟.

2.2 主目录的组织结构与共享数据识别方法

通过主从目录结构可以有效降低访问私有数据的延迟,但是传统的硬件实现存在可扩展性的问题,并且硬件层面难以实现共享与私有数据的严格划分,已有的方法多是软硬件协同实现[7].针对上述问题,本节提出了高可扩展的主目录缓存结构,主目录占用的存储空间随处理器核数的增加呈对数函数增长,可扩展性好,并基于主目录实现了对软件透明的共享数据识别方法.

在主从目录结构设计中,主目录用来记录私有数据副本在私有缓存的分布情况,其组织结构如图5所示.在图5中,末级缓存根据缓存数据的地址分成多个块(bank),每个块内部采用多路组相联结构,每个缓存组具有多个缓存行(line).在主从目录结构设计中,主目录与LLC采用紧耦合设计作为LLC缓存行的一部分,其由2个域组成:私有数据标记(P/S)与私有数据副本指针(pointer).其中,私有数据标记用来区分该缓存行是否是私有数据.当数据为私有数据时,私有数据副本指针用来指向缓存该数据的处理器核;当数据为共享时,该指针用来指向数据在从目录中的位置.

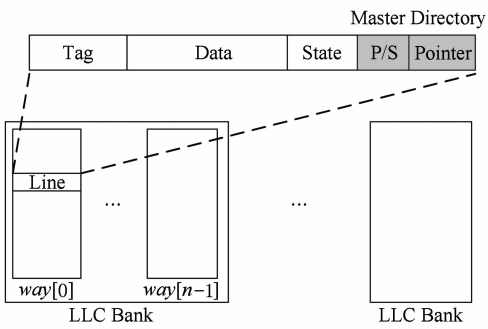


Fig. 5 Structure of master directory.

图5 主目录组织结构

在共享存储编程模型中,硬件模块难以准确地区分共享与私有数据.在主从目录结构设计中,为了区分共享与私有数据,本文提出了基于时间窗口的共享与私有数据划分方法,即在一定的时间范围内如果数据没有被多个处理器核共享,则可认为该数据在时间窗口范围内是私有的;否则认为该数据被共享.

在基于时间窗口的共享与私有数据划分方法中,私有状态与共享状态的转换如图6所示.时间窗口是指应用程序的数据从进入末级缓存开始,到离开末级缓存为止的一段可观测的时间区间.共享是指当前该数据在多个私有缓存中存在副本;私有是指当前该数据在私有缓存中最多只有一个副本.例如处理器核发起对地址 A 的数据访问,当末级缓存发生缺失时开启地址 A 的时间窗口,并设置对应数据的状态为私有状态.当末级缓存对地址 A 的缓存行执行替换时,该缓存行结束时间窗口并清空状态位.在时间窗口范围内,对私有数据的写操作或者来自同一个处理器核的读操作不会改变数据私有状态;对共享数据的读操作不会改变数据的共享状态;对私有数据执行读操作且发起方与上次访问该数据的处理器核不同,需要将数据状态改变为共享状态;对共享数据执行写操作或者记录该共享数据的从目录发生替换时,需要将数据改变为私有状态.

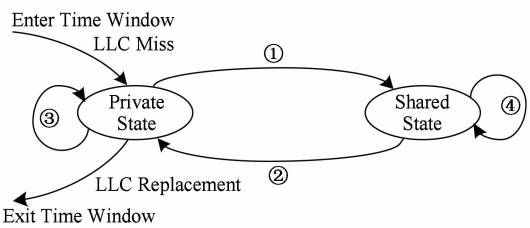


Fig. 6 State transition model in a timing window.

图6 时间窗口内数据的状态转换模型

2.3 从目录组织结构

在主从目录缓存结构设计中,从目录维护共享数据的一致性,用来记录共享数据在私有缓存中的分布情况.为了降低目录缓存实现的硬件开销,提高从目录设计的可扩展性,从目录采用有限位标签、多路组相联的稀疏目录结构,其组织结构如图7所示.其中,每个缓存组包含多个缓存行,每个缓存行包含3个域有限位标签(limited tag)、共享副本向量(shared vector)以及状态位(state).在访问从目录

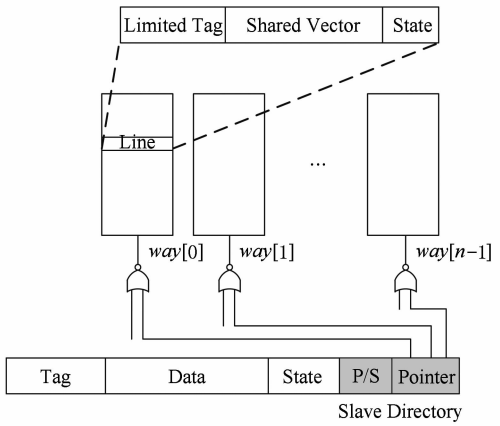


Fig. 7 Structure of slave directory.

图7 从目录组织结构

的过程中,有限位标签用来协助检查缓存行是否命中,根据系统配置的不同,有限位标签可以配置成不同的数据宽度(某些配置下可以为0);共享副本向量被用来记录共享数据副本在私有缓存中的分布情况;状态位被用来实现从目录的替换策略等.

从目录的查找过程(lookup)需要与主目录协同工作.当LLC中的缓存行处于共享状态时,主目录中的私有数据副本指针被用来指向哪一路从目录记录了该缓存行的共享向量.根据处理器核数(N)与从目录组相联度(W_s)的不同,从目录的查找过程可以分成2种情况:

1) 当 $N \geqslant W_s$ 时,私有数据副本指针的数据宽度($\mathrm{lb}\,N$)不小于区分从目录不同缓存路需要的数据宽度($\mathrm{lb}\,W_s$).在此情况下,主目录中的私有数据副本指针有能力区分从目录中同一组内的不同缓存行,因此从目录可以采用0-bit标签,即该域的数据宽度为0.在这种情况下,创建从目录项的过程需要将该目录项的从目录查找信息写入主目录的私有数据副本指针中;从目录没有查找过程,由主目录项的私有数据副本指针域直接指定;在从目录的替换过程中,需要置L1缓存的数据副本无效,并更新主目录信息.

2) 当 $N < W_s$ 时,私有数据副本指针的数据宽度($\mathrm{lb}\,N$)小于区分从目录不同缓存路需要的数据宽度($\mathrm{lb}\,W_s$).在此情况下,主目录中的私有数据副本指针不足以区分从目录中同一组内的不同缓存行,为此从目录采用($I+N$)-bit有限位标签.假设主目录与从目录分别包含 I_m 与 I_s 个缓存组.当 $I_m \leqslant I_s$ 时,不需要 I-bit域;当 $I_m > I_s$ 时,从目录 I-bit域保存了主目录的组信息,被用来区分来自主目录不同缓存组的访问.由于主目录的索引位比从目录的索引位长,且从目录索引位与主目录索引位的低位相同。因此为了减小存储开销,I-bit域仅缓存主目录索引位的高位,表示为 I_{offset},$I_{\mathrm{offset}} = I_m - I_s$.从目录 N-bit域保存了主目录的缓存路信息,被用来区分来自同一缓存组但不同缓存行的访问.

在这种情况下,在从目录项的创建过程中,需要保存创建该项的主目录项信息,包括缓存路信息与 I_{offset}.在从目录查找过程中,主目录发起访问请求,该请求由访问地址与发起访问的主目录缓存路信息组成,从目录根据地址找到相应的缓存组,用访问请求的内容与组内每个缓存行的有限位标签作比较,来判断缓存行是否命中.在从目录的替换过程中,需要置L1缓存的数据副本无效,并更新主目录信息.

图8展示了主目录中的不同缓存行访问从目录同一个缓存组的情况.在主目录中,缓存行 A(line-A)与缓存行 B(line-B)来自同一个缓存组 X,缓存行 C(line-C)来自另外的缓存组 Y.根据主目录与从目录间缓存组数量的不同,line-A,line-B 与 line-C 可能访问从目录的同一个缓存组.在示例中,从目录的缓存组 S 同时缓存了主目录的缓存行 A,B,C 的一致性共享信息.当主目录的缓存行 A 向从目录发起访问时,请求信息包括地址($Address$)与缓存路信息($way\,A$),从目录通过地址索引到相应的缓存组 S,将组内所有缓存行的标签与请求作比较,发现缓存行 $way[0]$-($I(A)$,$N(A)$)命中.

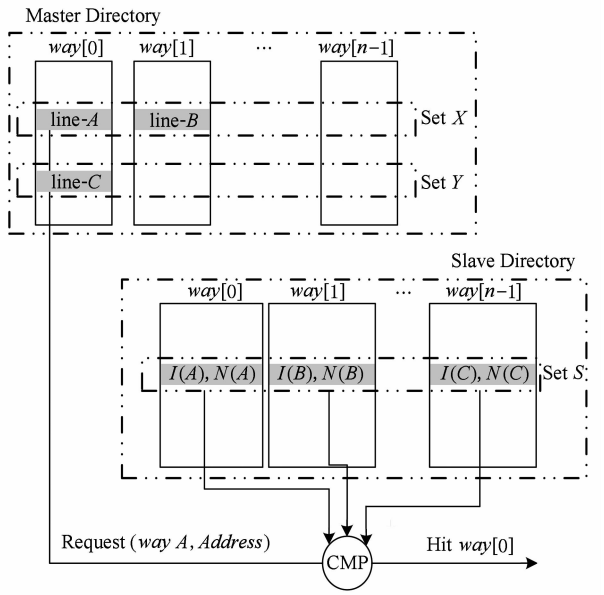


Fig. 8 Hit or miss of slave directory is judged by limited tags.

图8 通过有限位标签判断缓存行在从目录是否命中

商用处理器的参数配置,如Intel-Xeon-E5处理器,其采用46 b的系统地址空间;LLC采用20路组相联,容量20MB,LLC缓存行宽度是64 B.在从目录的结构设计中,根据上述配置,LLC的缓存组数量为 $I_{\mathrm{LLC}}=2^{14}$,LLC的标签位宽度为 $width[\mathrm{tag_{LLC}}]=(46-14-6)\,\mathrm{b}=26\,\mathrm{b}$.当稀疏目录采用相同的缓存组数量($I_{\mathrm{LLC}} = I_{\mathrm{Dir}}$)时,从目录的标签位宽度为 $width[\mathrm{tag_{Dir}}]=26\,\mathrm{b}$,与保存副本分布情况的共享向量相比,目录中标签位的存储开销占的比重非常大.而本节提出的有限位标签的从目录结构设计可以有效降低从目录的硬件设计开销.

3 实验评估

本节主要介绍了实验平台,包括系统组织结构

以及实验结果的对比分析.实验主要从硬件开销、系统性能与访问缺失延迟等方面评估了3种目录实现方法,分别是缓存内目录、稀疏目录与本文提出的主从目录缓存结构.

3.1 实验平台

本文基于Simics[12]+GEMS[13]的组合实验平台进行模拟评估.其中,Simics是风河公司(Wind River)开发的全系统模拟器,可以模拟甲骨文公司的SPARC系列处理器,并能够启动Solaris操作系统.GEMS是威斯康辛大学(Wisconsin)开发的时序模型,对缓存结构、片上网络以及一致性协议等进行了详细的建模.

实验基于GEMS建立了多核处理器的时序模型.模型中的处理器结构、缓存的容量信息、延迟信息等系统参数配置以Oracle的UltraSPARC T1处理器[14]为基础,同时结合了洛桑联邦理工学院(EPFL)对可扩展处理器(scale-out processor)的研究成果[8],其组织结构如图9所示,系统参数配置如表1所示.表1中,0.25×,0.5×,1×与2×分别表示0.25倍、0.5倍、1倍与2倍.时序模型模拟了16个处理器核,每个处理器核配有各32 KB的私有L1数据(指令)缓存,所有处理器核逻辑上共享6 MB末级缓存,物理上共享的末级缓存分成4个存储块(每4个处理器核配一个存储块),组件之间通过片上的交叉开关相连.在访问延迟方面,L1为2个时钟周期,L2为8个时钟周期,目录缓存为5个时钟周期,交叉开关为4个时钟周期,内存控制器为100个时钟周期.

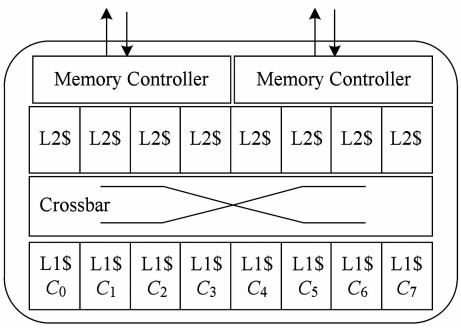


Fig. 9 Architecture of chip multi-processor.

图9 处理器的组织结构

Table 1 Parameters of System Configuration

表1 系统配置参数

Modules	Configurations
Processor	16 cores.
Core	SPARC instruction set, in-order-core, cache line 64 B.
L1-Cache	Private, 32 KB, 2-way set-association, 2-cycle access latency.
LLC	Shared, 6 MB, 12-way set-association, 8-cycle access latency.
Directory Cache	According to L1 capacity, the capacity of directory cache is divided into four categories: 0.25×, 0.5×, 1×, 2×. 5-cycle access latency, 4-way set-association.
Cache Coherence	Two-level MESI protocol, silent replacement.
Network	Crossbar, out-of-order, 4-cycle link latency.
Memory	100-cycle access latency.

本实验所使用的测试程序为大数据应用程序测试集CloudSuite-v1.0[15],该测试集是洛桑联邦理工学院为评估云数据中心服务器而设计开发的新型应用程序集合,如表2所示.CloudSuite测试集的程序来自真实的云数据中心应用,程序规模大,时序模拟时间长.为了缩短模拟时间,本实验采用了分段抽样取平均值的方法,每次抽样的执行过程分成3个阶段:1)从抽样点功能模拟1亿条指令;2)时序模拟1亿条指令,预热片上存储空间;3)统计1亿条指令的执行结果,将多次抽样的结果取平均值作为最终的系统行为输出.

Table 2 Benchmark Cloudsuite-v1.0

表2 Cloudsuite-v1.0测试程序

Programs	Characters
Cassandra	Data serving application. The program is a No-SQL database implementation with Apache Cassandra. It simulates a 15GB database servicing multiple clients.
Nutch	Web searching application. The program simulates the process of setting up Web index and searching based on Apache Nutch.
Classification	Map-reducing application. The program uses Bayesian algorithm to set up label for 4.5GB Web pages from Wiki.
Streaming	Media streaming application. The program simulates the server provide video for clients.
Cloud9	Software testing application. The program simulates the symbolic execution in Cloud9.

3.2 硬件开销

假设系统中有N个处理器核,缓存行宽度为L字节,每个处理器核具有容量为C_{L1}的数据(指令)缓存,所有处理器核共享容量为C_{LLC}的末级缓存,则末级缓存的缓存行数量为$B_{LLC}=C_{LLC}/L$,所有L1缓存的缓存行数量为$B_{L1}=(C_{L1\text{-}D}+C_{L1\text{-}I})\times N/L$.

在缓存内目录结构设计中,目录的容量主要受末级缓存容量的影响,其容量的计算公式为$B_{LLC}\times Entry_{In\text{-}Cache\text{-}Dir}$.其中,$Entry_{In\text{-}Cache\text{-}Dir}$为每一项的目录

开销,在表 1 所示的系统配置下 $Entry_{\text{In-Cache-Dir}}=18$ b,目录缓存的总容量为 216 KB.

在稀疏目录结构设计中,为了避免组相联度过大而引入的查找功耗过高,已有研究工作[6,16]发现通过增加稀疏目录每一路的缓存行数量可以避免同一缓存组内的访问冲突,尤其当稀疏目录的缓存行数量为所有 L1 缓存行总数的 2×时,可以将同一缓存组的访存冲突减小到合理的范围内. 如表 3 所示,本文根据稀疏目录中缓存行与 L1 缓存行总量的倍数关系,将稀疏目录分成了 0.25×,0.5×,1×与 2×这 4 种情况,并分别计算其容量的开销. 当比例为 R 时,稀疏目录的容量约为 $R\times B_{\text{L1}}\times Entry_{\text{Sparse-Dir}}$. 其中,$R$ 为稀疏目录缓存行与 $L1$ 缓存行的比例;$Entry_{\text{Sparse-Dir}}$ 由标签、共享副本向量与状态 3 部分组成,在表 1 所示的系统配置下 $Entry_{\text{Sparse-Dir}}\approx 46$ b.

Table 3 Capacity of Directory Cache in Different Configurations

表 3 不同比例关系下的目录缓存容量 KB

Directory Cache Design		Capacity of Directory Cache			
		0.25×	0.5×	1×	2×
In-Cache Directory		216	216	216	216
Sparse Directory		20.5	41	82	164
Master-Slave Directory	Master	60	60	60	60
	Slave	8	16	32	64
	Total	68	76	92	124

在主从目录缓存结构中,主目录采用是缓存内目录结构,故主目录的容量主要与末级缓存的缓存行数量相关,其容量的计算公式为 $B_{\text{LLC}}\times Entry_{\text{Master-Dir}}$,其中 $Entry_{\text{Master-Dir}}=\text{lb}\ N+1$. 在表 1 所示的系统配置下 $Entry_{\text{Master-Dir}}=5$ b,主目录的总容量为 60 KB. 从目录采用有限位标签的稀疏目录结构,在不同的倍数关系下,从目录的容量为 $R\times B_{\text{L1}}\times Entry_{\text{Slave-Dir}}$,其中 $Entry_{\text{Salve-Dir}}$ 由有限位标签、共享副本向量与状态 3 部分组成,在表 1 所示的系统配置下有限位标签长度为 0,故 $Entry_{\text{Slave-Dir}}=18$ b,不同配置下的从目录缓存容量如表 3 所示.

研究工作[6,16]指出,稀疏目录在 2×比例关系下可以将目录缓存导致的性能损耗控制在较低的水平. 在 2×的比例关系,通过表 3 可以看出主从目录缓存具有较低的硬件开销. 在接下来的实验评估中,主要分析在 2×比例关系下不同目录结构设计的性能参数.

3.3 数据访问延迟

如引言所述,数据访问延迟是衡量面向大数据应用数据中心的关键参数. 本节分析对比了 3 种目录设计结构的平均数据访问延迟与最大数据访问延迟.

图 10 展示了在 2×的比例关系下缓存内目录、稀疏目录以及主从目录缓存的 L1 缓存缺失的平均访问延迟. 从图 10 可以看出,缓存内目录平均延迟开销最小;稀疏目录的平均数据访问延迟最大,增加了 46.94%的延迟开销,这部分延迟主要是由于增加了对稀疏目录的访问以及访问所引入的网络延迟;而本文提出的主从目录缓存以最小的硬件开销取得了与缓存内目录接近的平均数据访问延迟. 与缓存内目录相比,主从目录缓存虽然增加了 5.14%的数据访问延迟,但是降低了 42.59%的硬件开销;与稀疏目录相比,主从目录缓存不仅降低 24.39%的硬件开销,同时降低了 28.45%的数据访问延迟.

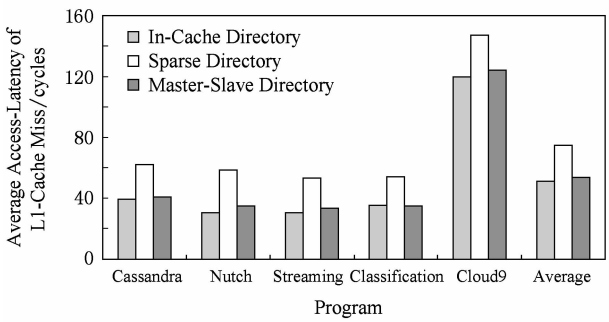


Fig. 10 Average miss-latency of L1-cache with 2× capacity of directory cache.

图 10 在 2×比例关系下 L1 缓存缺失的平均数据访问延迟

在图 10 中,大部分应用的平均数据访问延迟在 30~65 个时钟周期之间,而应用 Cloud9 的平均数据访问延迟超过了 100 个时钟周期. 该现象是由于 Cloud9 的 LLC 缺失率非常高造成的,其他 4 类应用的 LLC 缺失率均在 20%以下,而 Cloud9 的 LLC 缺失率超过了 80%.

虽然主从目录缓存取得了硬件开销与平均数据访问延迟较好的折中,然而,主从目录缓存增加了某些情况下的关键路径长度,导致主从目录缓存在最坏情况下的数据访问延迟增大. 图 11 展示了在 2×的比例关系下缓存内目录、稀疏目录以及主从目录的 L1 缓存缺失的最大数据访问延迟. 从图 11 可以看到,主从目录缓存的最坏情况平均访问延迟最大,分别为缓存内目录、稀疏目录的 2.3 倍、1.8 倍. 由于最坏情况下的访存延迟较大,因此主从目录缓存结构不适合有严格实时性约束类的应用.

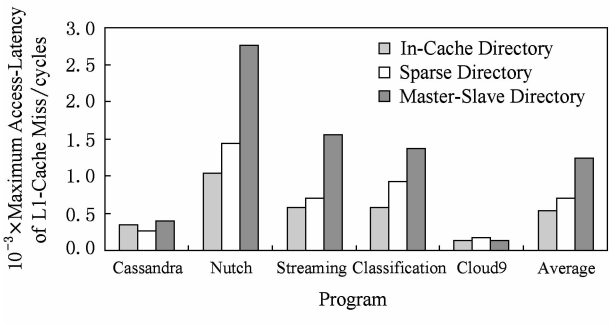


Fig. 11 Maxmium miss-latency of L1-cache with 2× capacity of directory cache.

图 11 在 2×比例关系下 L1 缓存缺失的最大数据访问延迟

如图 11 所示,在最大数据访问延迟方面,3 种不同的目录结构设计在不同的应用程序之间展现了不同的变化规律;最大数据访问延迟约为平均数据访问的几十倍,可以得知最大数据访问延迟是由网络拥塞而导致的,是程序行为与目录结构设计共同作用的结果,因此在不同的应用之间表现出了不同的变化规律.

对于处理器核之间的共享型数据访问(读后读、读后写、写后读),由于主从目录缓存需要主从目录协同工作,从而增加了最坏情况下数据块访问的阻塞时间,加剧了最坏情况下的网络拥塞,导致最坏情况下的数据访问延迟大.

虽然主从目录结构在最坏情况下的数据访问延迟大,但是由于末级缓存访问命中时的数据访问以私有数据为主(占总量的 80%,如图 12 所示),因此主从目录结构的平均数据访问延迟较小.

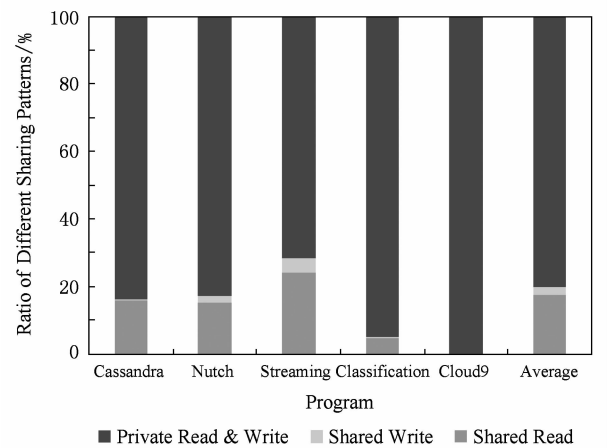


Fig. 12 Different sharing patterns of hit accessings in LLC.

图 12 LLC 命中时不同类型访问的比例

3.4 程序执行性能

在 2×的容量比例关系下,图 13 通过单位时钟周期内的执行指令数(instruction per cycle, *IPC*)对比了 3 种目录结构设计在执行不同应用时的性能. 如图 13 所示,在执行所有的应用程序中,缓存内目录结构均取得了最高的 *IPC* 值,2×容量的稀疏目录结构均取得最低的 *IPC* 值. 在执行应用程序 Cassandra,Classification 与 Cloud9 时,主从目录缓存与缓存内目录一起取得了最高的 *IPC* 值;在执行应用程序 Nutch 与 Streaming 时,主从目录缓存结构取得了仅次于缓存内目录的 *IPC* 值.

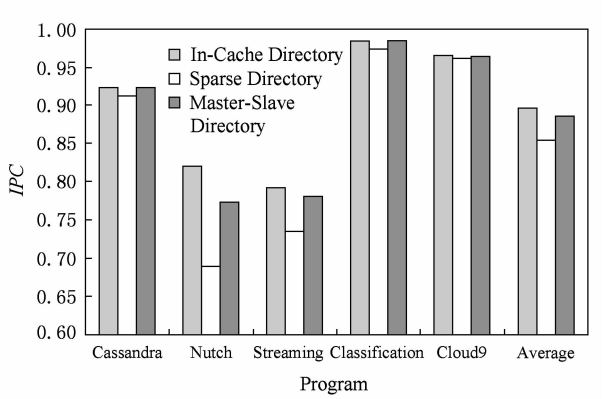


Fig. 13 *IPC* of programs with 2× capacity of directory cache.

图 13 在 2×比例关系下程序执行的 *IPC*

从图 13 统计的平均值来看,缓存内目录结构取得了最高的 *IPC* 值,但是其硬件开销大、目录的容量利用率低;与之相比,主从目录缓存结构仅损失了 1.1%的整体性能,但是降低了 42.59%的硬件开销;与 2×容量比例的稀疏目录相比,不仅减小了 24.39%的硬件开销,同时提升了 3.5%的执行性能.

3.5 从目录对容量的敏感度分析

图 14 对比了不同应用在 4 种比例关系下的从目录替换率(2×,1×,0.5×,0.25×),并以 0.25×

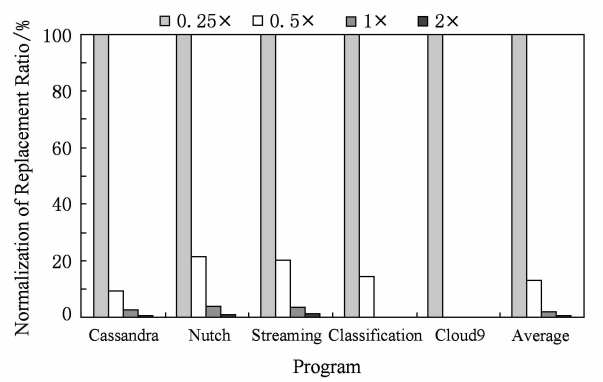


Fig. 14 Replacement ratio under different directory cache capacity.

图 14 从目录容量对从目录替换率的影响

比例关系作为基准进行了归一化处理.从图 14 可以看出,从目录容量在 2×到 0.5×的区间内,从目录的替换率没有明显的升高;当目录容量降低到 0.25×时,从目录的替换率快速上升,在所有测试的应用程序中最少增加 79%,平均增加 87%.从硬件开销、目录替换率的角度考虑,0.5×的比例关系适合从目录的容量设计.

图 15 分别对比了 0.5×容量主从目录缓存与 2×容量稀疏目录的 L1 缓存缺失率与 *IPC*.从图 15 可以看出,与稀疏目录结构相比,主从目录缓存结构取得了更低的 L1 缓存缺失率以及更高的 *IPC* 值,提升了系统性能.

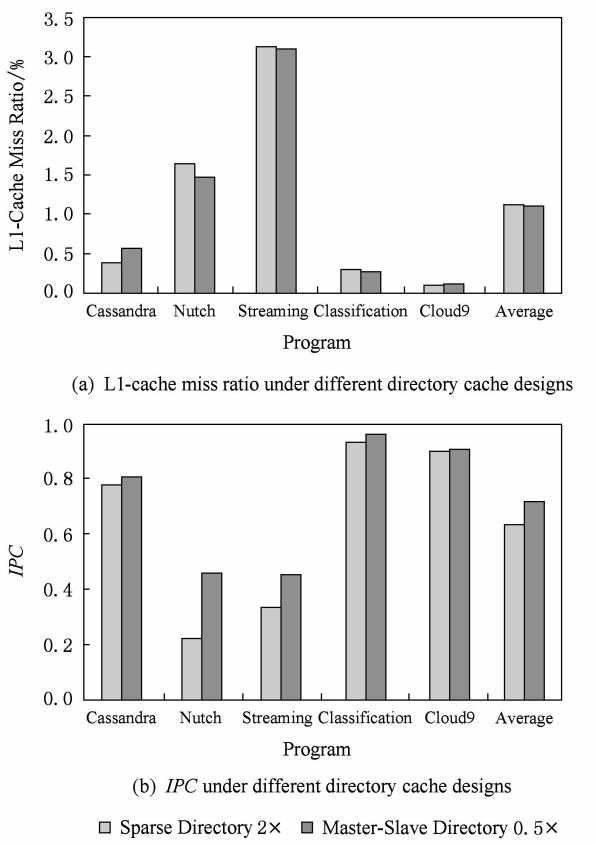


Fig. 15 L1-cache miss ratio and *IPC* of programs for sparse directory 2× and master-slave directory 0.5×.

图 15 对比 0.5×容量主从目录缓存与 2×容量稀疏目录的 L1 缺失率与处理器 *IPC*

从图 15(a)看出,在 L1 缓存缺失率方面,由于主从目录缓存的容量利用率更高,产生更少的使无效消息,使得主从目录缓存几乎在所有的程序中取得了更低的缺失率,除了程序 Cassandra 以外.针对 Cassandra 程序,程序中对共享数据的访问相对集中.而主从目录缓存结构中,主目录与从目录分别维护共享与私有数据的缓存一致性,当共享数据访问集中在部分缓存组时,该结构设计没有充分利用末级缓存的大容量与高组相联度的优势,并且从目录的小容量不利于在缓存组之间分散数据访问,导致了在执行 Cassandra 程序时 L1 缓存缺失率更高.

在 *IPC* 方面,*IPC* 是由多种因素共同作用的结果,以访存操作的 *IPC* 为例,*IPC* 是由 L1 缓存的缺失率与缓存缺失的数据访问延迟共同决定的.主从目录缓存结构不仅在 L1 缓存缺失率方面有所降低,并且降低了缓存缺失的数据访问延迟,因此 0.5×容量主从目录缓存结构与 2×容量稀疏目录相比取得了更高的 *IPC* 值,如图 15(b)所示.

4 相关研究工作

为了解决稀疏目录中访问冲突产生的替换问题,提高稀疏目录的容量利用率,降低硬件实现开销,研究人员基于稀疏目录提出了大量的新型目录缓存结构,下面介绍 6 种有代表性的研究工作.

文献[6]提出的 Cuckoo Directory 是一种提高目录缓存容量利用率的方法,Cuckoo Directory 基于 Hash 函数采用迭代插入的方法模拟了一个更大组相联度的稀疏目录结构.该结构大大提高了目录缓存的容量利用率,但是迭代插入的过程进一步增加了处理器的访存延迟与功耗.文献[7]提出 Deactivating Directory,有效提高目录容量的利用率.该结构通过软件标识数据是否被共享,并基于共享信息过滤私有数据,仅维护共享数据的一致性.文献[16]提出的 Stash Directory 是一种推测执行方法.Stash Directory 在目录缓存发生替换时并非立即将私有缓存的数据副本设置为无效,而是延迟该过程来避免 L1 缓存的抖动问题.文献[17]发现真实系统中一段连续的存储区域具有相同的数据访问特性,并基于该现象提出新型目录结构,该结构针对这种连续的存储区域使用一个目录项进行记录,避免目录缓存的容量浪费.上述研究工作重点解决的是目录缓存的冲突替换问题,与之相比,本文提出的方法面向大数据中延迟敏感类的应用,重点解决数据访问延迟的问题,主从目录结构在保证较低冲突替换率的同时,更重要的是降低了处理器的访存延迟与目录缓存的硬件开销.

文献[18]提出的 Tagless Directory 是复制标签目录的变种,可以有效降低实现目录缓存的硬件

开销. Tagless Directory 基于布隆过滤器(Bloom filter)将同一个处理器核的同一缓存组内缓存行标签保存在一个二进制向量中. Tagless Directory 避免了存储标签位的开销,提高了目录缓存的存储效率,但是与复制标签目录相同,Tagless Directory 的查找功耗随着处理器核数增长而增加,导致其可扩展性不好. 与 Tagless Directory 相比,本文提出的有限位标签目录缓存不仅避免了缓存完成标签引入存储开销,同时避免了过高的查找功耗,具有更好的扩展性.

针对大数据应用,文献[19]提出了一种面向大数据应用的目录结构设计. 该研究发现在大数据应用程序中一段连续的地址空间往往具有相同的一致性属性,因此可以用一个目录项表示一段连续的存储空间. 通过大数据测试程序评估,该研究能够有效降低目录的存储开销. 与之相比,本文更多关注面向大数据应用时处理器的访存延迟.

5 结 论

目前数据中心的核心负载大数据应用程序是时延敏感型应用,而稀疏目录结构作为多核处理器结构中维护缓存一致性的关键部件,其维护一致性的过程增大了数据访问延迟. 为了克服稀疏目录导致数据访问延迟增加的问题,本文针对大数据应用以私有数据访问为主的特点,提出了新型主从目录缓存结构. 实验结果表明本文提出的方法可以有效降低片上多核处理器的平均数据访问延迟,在处理器访存延迟、目录缓存的硬件开销以及目录缓存的容量利用率方面取得较好的折中. 但是本文的方法增大了目录缓存在最坏情况下的关键路径长度,因此主从目录缓存结构不适合有严格实时性约束类的应用. 此外,在本文的实验中发现,商用一致性协议存在的静默降级(silent replacement)策略[2]会导致目录缓存中产生伪副本记录,这些伪副本记录在私有缓存中没有数据副本,但是却占用目录缓存的容量. 下一步工作中,我们将继续研究伪副本记录的消除以及降低最坏情况下关键路径长度的问题.

参 考 文 献

[1] Ferdman M, Adileh A, Kocberber O, et al. Clearing the clouds: A study of emerging scale-out workloads on modern hardware [C] //Proc of the 17th Conf on Architecture Support for Programming Languages and Operating Systems (ASPLOS). New York: ACM, 2012: 37-48

[2] Sorin D J, Hill M D, Wood D A. A Primer on Memory Consistency and Cache Coherence [M]. San Rafael, CA: Morgan & Claypool Publishers, 2011

[3] Barroso L A, Gharachorloo K, McNamara R, et al. Piranha: A scalable architecture based on single-chip multiprocessing [C] //Proc of the 27th Annual Int Symp on Computer Architecture (ISCA). New York: ACM, 2000: 282-293

[4] Sun. OpenSPARC™ T2 core microarchitecture specification [R/OL]. Sun MicroSysmtes, Inc, 2007 [2015-04-20]. http://www.oracle.com/technetwork/systems/opensparc/t2-06-opensparct2-core-microarch-1537749.html

[5] Singhal R. Inside Intel next generation Nehalem micorarchitecture [R/OL]. Intel Corporation, 2008 [2015-04-20]. http://weblab.cs.uml.edu/~bill/cs515/Intel_Nehalem_Processor.pdf

[6] Ferdman M, Pejman L K, Balet K, et al. Cuckoo directory: A scalable directory for many-core systems [C] //Proc of the 17th Int Symp on High Performance Computer Architecture (HPCA). New York: ACM, 2011: 169-180

[7] Cuesta B A, Ros A, Gomez M E, et al. Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks [C] //Proc of the 38th Annual Int Symp on Computer Architecture (ISCA). New York: ACM, 2011: 93-104

[8] Pejman L K, Grot B, Fredman M, et al. Scale-out Processors [C] //Proc of the 39th Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2012: 500-511

[9] Gupta A, Weber W, Mowry T. Reducing memory and traffic requirements for scalable directory-based cache coherence schemes [C] //Proc of the 1990 Int Conf on Parallel Processing (ICPP). Berlin: Springer, 1992: 167-192

[10] Martin M M K, Hill M D, Sorin D J. Why on-chip cache coherence is here to stay [J]. Communications of the ACM, 2012, 55(7): 78-89

[11] Novakovic S, Daglis A, Bugnion E, et al. Scale-out NUMA [C] //Proc of the 19th Int Conf on Architecture Support for Orogramming Languages and Operating Systems (ASPLOS). New York: ACM, 2014: 3-18

[12] Magnusson P S, Christensson M, Eskilson J, et al. Simics: A full system simulation platform [J]. IEEE Computer, 2002, 35(2): 50-58

[13] Martin M M K, Sorin D J, Beckmann B M, et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset [J]. Computer Architecture News, 2005, 33(4): 92-99

[14] Bryg W, Alabado J. The UltraSPARC T1 processor-high bandwidth for throughput computing [R/OL]. Sun MicroSysmtes, Inc, 2005 [2015-04-16]. http://ftp.ocs.ru/sun/Products/Systems/Niagara/INTEGRATOR/UST1_bw_v1.0.pdf

[15] Adileh A, Kaynak C, Pejman L K, et al. Tutorial: CloudSuite on Flexus [R/OL]. Swiss Federal Institute of Technology in Lausanne, 2013 [2015-04-20]. http://parsa.epfl.ch/cloudsuite/docs/CloudSuite2.0-on-Flexus-isca13.pdf

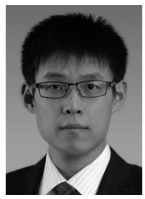
[16] Demetriades S, Sangyeun C. Stash directory: A scalable directory for many-core coherence [C] //Proc of the 20th Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2014: 177-188

[17] Fang Lei, Liu Peng, Hu Qi, et al. Building expressive, area-efficient coherence directories [C] //Proc of the 22nd Int Conf on Parallel Architectures and Compilation Techniques (PACT). Piscataway, NJ: IEEE, 2013: 299-308

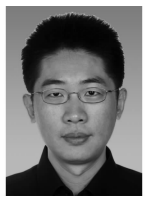
[18] Zebchuk J, Srinivasan V, Moshovos A. A tagless coherence directory [C] //Proc of the 42nd Annual Int Symp on Microarchitecture (MICRO). New York: ACM, 2009: 423-434

[19] Zebchuk J, Falsafi B, Moshovos A. Multi-grain coherence directories [C] //Proc of the 46th Annual Int Symp on Microarchitecture (MICRO). New York: ACM, 2013: 359-370

Wang Endong, born in 1966. MS, professor. Fellow member of China Computer Federation. His main research interests include computer architecture, hybrid storage system and interconnect protocols for scalable system.

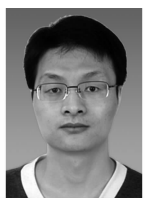


Tang Shibin, born in 1986. PhD. His main research interests include computer architecture, cache coherence and on-chip memory system.



Chen Jicheng, born in 1976. PhD, associate professor. Member of China Computer Federation. His main research interests include computer architecture, cache coherence and integrated circuit design (chenjch@inspur.com).

Wang Hongwei, born in 1982. PhD. His main research interests include computer architecture, high performance computing and artificial intelligence (wanghongwei@inspur.com).



Ni Fan, born in 1984. PhD. His main research interests include computer architecture, real time system and operating system design (nifan@inspur.com).

Zhao Yaqian, born in 1981. PhD. Her main research interests include computer architecture and machine learning (zhaoyaqian@inspur.com).