多核处理器 Cache 一致性关键问题与研究摘述

摘要:介绍了片上多核处理器一致性问题的由来.总结了多核时代高速缓存一致性协议设计的关键问题,综述了国内外近年来关于众核处理器 cache 一致性的相关研究.

关键词:cache 一致性协议;片上多核处理器

Abstract: In this paper, the origin of cache coherence is carefully described. Further, the paper summarizes the key issue of cache coherence and reviews the study in this field a decade after entering the mulit-core era. At last, summed up many-core processor cache coherence related research work.

Key words: cache coherence protocol; chip multi-processor

1 Cache 一致性问题的起源与发展

1.1 Cache 一致性问题的由来

在多核处理器中,处理器核心对本地节点高速缓存的访问时延最短,对非本地节点高速缓存的访问时延则会受到片上互连结构和相对位置的影响.随着片上节点数量的增大,通信延时的增长,本地和非本地高速缓存访问延时的差异愈加明显.为了平衡这种时延,在多核处理器设计中引入了软\硬件共享存储模型,即,每个核都拥有私有一级缓存和共享的末级缓存(last level cache,简称 LLC).为了缩短数据访问片上网络(network on chip,简称 NoC)的延迟,

处理器核心获取所需数据之后在本地创建数据副本,而不论其他节点的高速缓存是否存在相同的数据.私有缓存机制保证了本地节点内的处理器核心独占本地节点的高速缓存资源.由此产生了缓存一致性(cache coherence)[1]问题,其根本原因是:在一个多处理器系统中,不同的缓存中可能存在同一个数据的多个副本.这些副本的存在,严重影响了程序执行的正确性.这样就需要缓存一致性协议(cache coherence protocol)来管理共享数据的多个副本.

缓存一致性问题有多种定义方式,Gharachorloo 等学者给出的定义[2]为:

(1)每次写操作对所有的核可见;

(2)写操作是顺序化的,即,所有的内核观察到相同访存序列.因此,缓存一致性要求写操作必须最终广播到参与的全部处理器中.同时,要求参与的内核所观察到的对同一个地址的写操作,必须按照相同顺序进行.Hennessy 和 Patterson 认为,缓存一致性确定了读取操作可能返回什么值,给出的定义如下所示[3].缓存一致性由 3 个不变式组成.

(1)处理器 P 读取位置 X,在此之前是 P 对 X 进行写入,在 P 执行的这一写入与读取操作之间,没有其他处理器对位置 X 执行写入操作,此读取操作总是返回 P 写入的值;

(2)一个处理器向位置 X 执行写入操作之后,另一个处理器读取该位置,如果读写操作的间隔时间足够长,而且在两次访问之间没有其他处理器向 X 写入,则该读取操作将返回写入值;

(3)对同一个位置执行的写入操作被串行化.一致性协议可以使用软件或者硬件方式来实现.在片上多核处理器中,缓存一致性协议的实现与片上网络密切相关.根据维护缓存块状态方法的不同,处理器可以使用侦听(snooping)和目录(directory)两大类机制.

1.2 侦听一致性协议

侦听一致性协议是利用总线广播(broadcast)机制来实现的,是 Cache 一致性协议最早的实现方式.系统中的所有高速缓存控制器都需要侦听系统中的一致性消息,以此来确定是否有一致性请求.最为经典的总线监听协议是 MESI[4],由 James Goodman 提出,在 x86,ARM 和 Power 处理器得以具体实现.

图 1 是广播一致性协议的简单示意图.在广播协议中,当处理器 P1 发生写(W)操作时,一个无效请求(虚线)被发送到所有其他的处理器,以获得适当的权限来执行写操作.接着写操作被执行.因为广播协议在执行时,占据了整个总线,P3 的读操作必须推迟到写操作完成才能进行.当 P3 发出读请求时,所有其他处理器必须侦听,直到数据返回给 P3 读操作完成.总的来说,对于规模较小的多核处理器基于广播的方法是适用的.

在侦听协议中,总线是所有一致性消息的定序点.所有连接到总线上的节点都以相同的顺序观察到总线上的一致性消息.但是同一时刻只能有一个消息在总线上传输,每个缓存控制器分别管理自身数据块的状态,并通过总线进行不同缓存间的状态同步.然而,随着多核规模的不断扩大,多个通信组件必将产生对总线的争用.加之进行作废\更新(invalidate\update)操作时会进行全系统的消息广播,因此,一致性产生的片上网络负荷会不断加重,同时伴随的功耗问题也不容忽视.因此,基于侦听的一致性协议可扩展性较差.

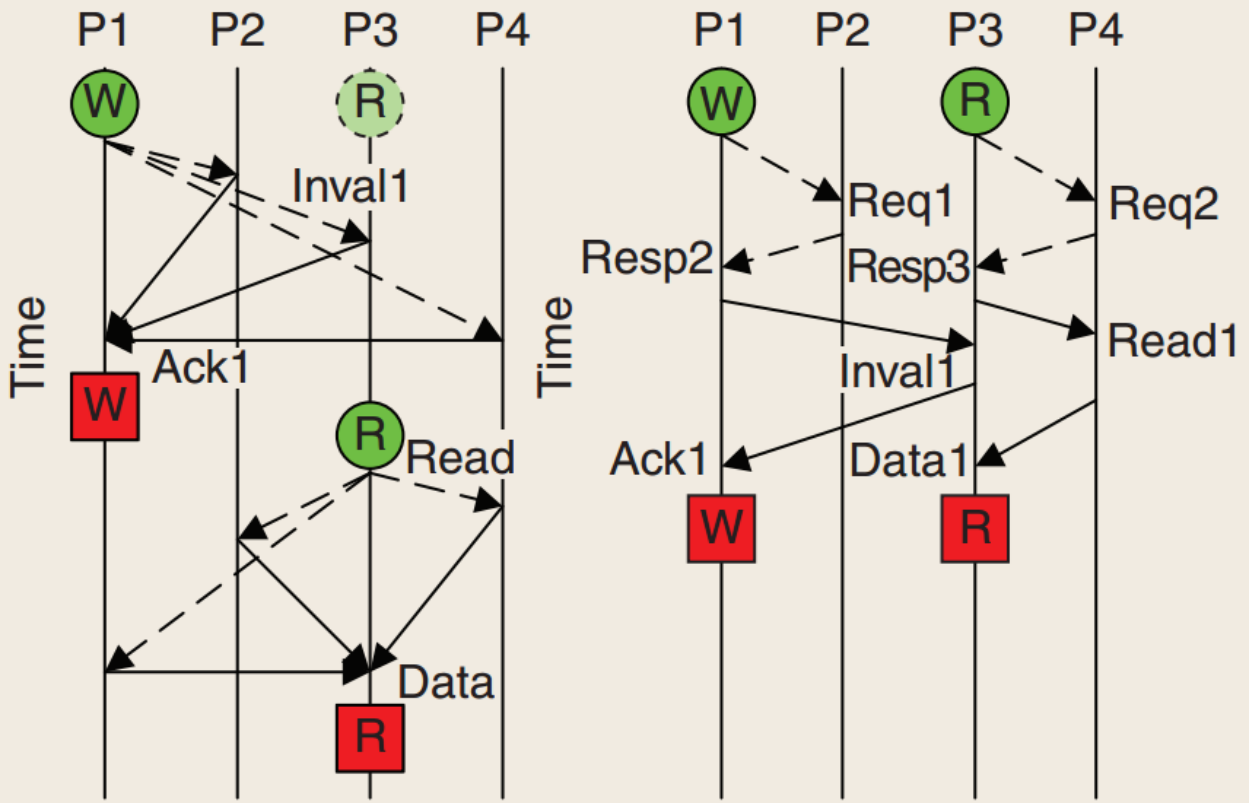

图 1 广播一致性协议和目录一致性协议示意图

1.3 目录一致性协议

目录一致性协议使用目录来全局记录缓存状态,包括数据的共享者列表、一致性状态等.在目录一致性协议中,目录充当了定序点的角色.消息的传输是以点对点的方式进行的,即,所有一致性消息通过一个目录结构来转发处理,因此,它可以有效地降低一致性消息对网络带宽的需求.图 1(b)是目录一致性协议的简单示意图.当 P1 发生写操作时,首先查询 Home 节点(P2)来确定当前所有者/共享者.Home 节点响应请求,P1 的无效请求接着被发送给当前所有者/共享者.每个相关节点都要给予回复确认.一旦 P1 已经收到了所有的应答,然后执行写操作.类似的过程,P3 完成读操作.在这种情况下,因为网络不是完全被占用,读和写都可以并行执行.目录一致性适用于弱一致性模型和大型系统.目前,商用的有 Intel 的 Corei7 系列处理器,它采用了 QPI(quick path interconnect)技术[5,6],支持 MESIF[7]一致性协议.

目录协议避免了广播消息,减少了完成一致性请求所需的通信量,同时避免了对顺序化互连结构的依赖,具有较好的可扩展性.目前,绝大多数片上多核处理器都采用基于目录的一致性协议.目录协议最基本的硬件实现方式是为存储器的每个数据块分配一个目录条目(entry),记录下该数据在哪些缓存有副本.在一个典型的目录条目中,包含了缓存块的状态、共享者列表、拥有者标识等信息.其中,共享者列表常采用位图的形式来实现,而拥有者标识则表示相应的处理器核标识或者节点标识号.如图 2 所示:目录条目{1000}M 显示,数据块 A 位于 Core0 的私有缓存中,状态为 Modified;目录条目{0110}S 显示,数据块 B 在 Core1 和 Core2 的私有缓存中都有副本,状态为 Shared.

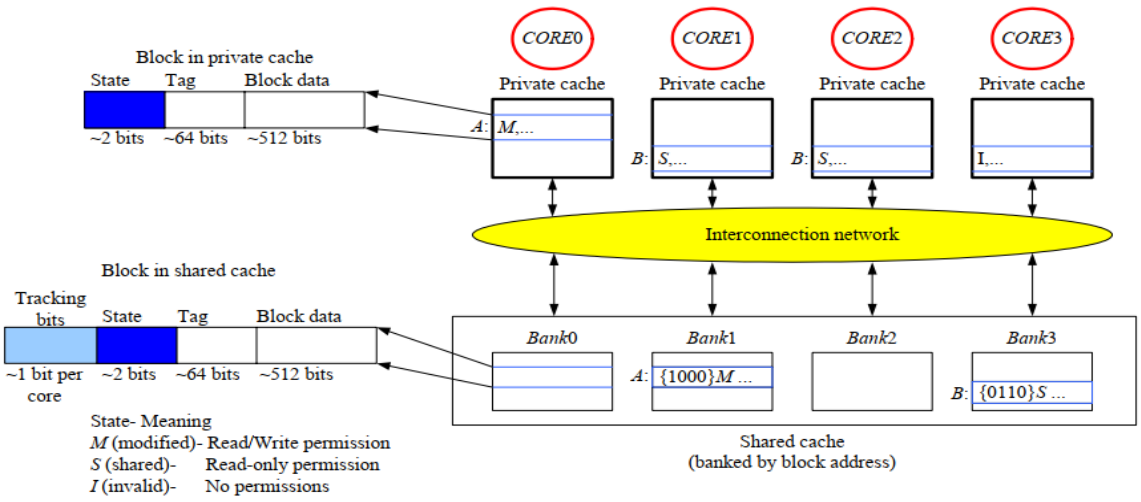

图 2 基于目录一致性协议的目录结构

1.4 经典的一致性协议

MESI 协议也称为 Illinois MESI 协议,是一种非常典型的缓存一致性协议[8].协议以缓存块的 4 种状态命名.

•M 代表修改状态,表示已修改过,当前缓存行中包含的数据与存储器中的数据不一致,只有该缓存中有最新的拷贝,其他缓存中无该块的有效拷贝;

•E 表示独占状态,说明只有一个处理器核有该内存块的缓存,并且该块在缓存中未被修改过,主存中是最新的;

•S 代表共享状态,表明该块在缓存中未被修改过,并且其他核存在该块的拷贝;

•I 代表无效状态,说明该块在缓存中是无效的,或者该块还没有进入缓存.

随着片上多处理器的出现,一些适合于新的计算机体系结构的缓存一致性协议相继提出,这些协议通常需要考虑存储系统的互联网络的特点.MESI 协议出现的变形,如 MOESI 协议和 MESIF 协议.AMD 公司的 MOESI 协议用于保证其超传输(Hyper Transport)互联系统的缓存一致性[9,10].MOESI 协议中,M,E,I 这 3 种状态与 MESI 协议相同,重新定义了 S 状态,

引入了一个新状态 O.状态 O 表示拥有者,在当前缓存行中包含的数据是最新的,而且在其他 CPU 中一定具有该 Cache 行的拷贝,其他拥有副本的缓存状态为 S.有且仅有一个 Cache 行状态为 O.在 MOESI 协议中,状态为 S 的缓存行中的数据与存储器中的数据不一定一致.如果在其他缓存中不存在状态为 O 的拷贝,则该缓存行中的数据与存储器一致;如果在其他缓存中存在状态为 O 的拷贝,则缓存行中的数据与存储器不一致.

Intel 提出了 MESI 协议的变种,即 MESIF 协议.该协议与 MOESI 协议类似,都是对 MESI 协议的扩充,因此比 MESI 协议更复杂.MESIF 协议解决的主要问题是 ccNUMA 架构中 SMP 子系统与 SMP 子系统之间缓存一致性问题[11].MESIF 协议引入了一个 F(forward)状态.在采用 ccNUMA 的处理器系统中,多个处理器的缓存中可能存在相同数据的拷贝,在这些拷贝中,只有一个缓存行的状态为 F,其他缓存行的状态都为 S.当缓存行的状态位为 F 时,缓存中的数据与存储器一致.当一个数据请求读取这个数据副本时,只有状态为 F 的缓存行可以将数据副本转发给数据请求者,而状态位为 S 的缓存不能转发数据副本.MESIF 协议有效地解决了 ccNUMA 处理器结构中,所有状态位为 S 的缓存同时转发数据副本给数据请求者而造成的网络拥塞问题.

2 研究的关键问题

多核 Cache 一致性研究涉及许多方面,其中要解决的关键问题可归纳如下:程序访存行为分析、目录组织结构、一致性协议流量(traffic)、一致性粒度(granularity)、目录协议的可扩展性(scalability)、低功耗以及具体软/硬件实现等.

2.1 程序访存行为分析

应用程序的访存行为是 Cache 一致性协议优化的根本依据.在单核时代,程序访存局域性理论对 Cache 的优化做出了重大的支撑.在多核时代,随着多线程程序的并行编程模型的出现,程序的并行度不断提高,这使得程序的局域性特征呈现出新的特点.比如:瑞士洛桑联

邦理工学院的学者 Alisafaee 提出了时空一致性跟踪(spatiotemporalcoherencetracking,简称 SCT)[12]方法.深入分析程序的访存行为,还可以发掘多种数据的共享模式(sharingpattern).利用软/硬件手段预测和识别这些共享模式,能够对一致性策略做出自适应的调整[13].目前,虽然对数据共享模式的研究在编译优化层面已经非常深入,但是该研究被应用到 Cache 一致性协议的优化中才刚刚开始.无论是在预测的精度还是为此付出的代价上都还有待进一步的优化,这也是近期研究的热点.

2.2 目录结构组织

目录能够精确记录跟踪共享数据信息,是当前多核处理器的标准解决方案,目录的组织结构与一致性协议密切相关.目录结构上的一些细微改动,就能够很大程度上降低目录的存储空间.每种目录组织结构都对应着在其上构建的一致性操作算法.目录组织结构设计的难点在于:用尽可能少的存储开销精确记录数据的共享信息,而不衰减系统性能.

2.3 一致性粒度

传统目录缓存的每个条目管理一个 Cache 行,在目前的 Cache 结构中,一个 Cache 行一般 64 个字节.而调查发现:在程序中绝大多数的数据是私有的,共享数据只占很少比例.对某些程序而言,多个并发线程可能会访问由多个 Cache 行组成的一个连续的“区域”.考虑到这个区域中的缓存块可能有相同的特性,可以将这个区域识别为一致性区域,为这个区域设置一个目录条目.一旦区域中发生私有/共享的变化,就会导致整个区域的瓦解,而不得不更新为 Cache 行的粒度.通过操作系统在运行时发掘共享数据信息,最小的粒度为一个页面,一般为 4KB~8KB,这是目前一致性研究中最大的一致性粒度.罗切斯特大学的 Zhao 等人使用缓存缺失率、作废率、数据的使用率这 3 个指标,分析了应用程序的行为特征.应用程序空间局域性和存储粒度之间的错误匹配,导致了相当大的未使用数据的移动;而应用程序共享粒度和一致性粒度之间的错误匹配,导致了伪共享(falsesharing),从而带来性能和带宽的惩罚[14,15].

2.4 一致性协议流量

在片上多核处理器中,一致性消息通过片上网络进行传输.因此,一致性协议不仅决定了共享存储系统内部如何通信,还影响到存储系统与内核、片上网络之间进行数据的传输[16].比如 Ros 提出的直接一致性协议,以一致性协议的动作本身为优化对象,减少了一致性消息操作的跳数[17].然而,在当前的片上网络研究中,往往以提高片上网络的吞吐率为目的.在仿真中,并没有真正考虑一致性消息传输的问题[18].因此,结合片上网络来进行的一致性模型优化,是提高一致性协议及片上网络性能的关键方法.

2.5 可扩展性

影响一致性协议可扩展性的因素主要有目录的存储开销、一致性交互延迟以及一致性操作动作等,不仅涉及到协议本身,而且与片上网络的性能也紧密相关.随着多核规模的扩大,层次化的多核层次化分组结构是必由之路.在分组内采用广播方式,在组间采用目录方式,这种混合的一致性协议已经开始在逐步探索之中.

2.6 低功耗

在目前的多核研究中,一方面降低功耗,一方面使程序的并行性得到发挥,这样仍能提高系统的整体性能.Cache 的功耗已成为当前多核体系结构设计面临的最大问题之一.Cache 的功耗分为静态功耗和动态功耗.

•静态功耗与其容量大小正相关,即:Cache 容量越大,功耗越大.而且随着 Cache 容量的增加,访问延时也迅速增加;

•动态功耗与程序的访存行为有密切关系,减少不必要的一致性访问的次数和降低每次读写访问的功耗,都能降低 Cache 的功耗.Cache 功耗的高低,已经成为评价 Cache 一致性协议优化的重要指标之一.

3 相关研究

技术的发展驱使单个芯片上集成的处理器核个数越来越多,各大半导体公司于 2006 年之后纷纷推出其众核处理器产品.2006 年,IBM 公司推出其 1025 核心的 Kilocore 众核处理器;Tilera 公司于 2007 年发布了其 64 核心的 TILE64,于 2009 年推出最新的 100 核心的 TILE-Gx100 众核处理器;Intel 公司于 2008 年披露了其 80 核心的 POLARIS 原型;Clear Speed 公司在 2008 年推出其 192 核心的 CSX700 处理器.即便如此,众核处理器的很多设计问题至今依然没有得到有效解决.由于众核处理器的处理器核心数目巨大,维护处理器核心之间的数据一致性呈现出新的需求,cache 一致性协议就成为亟待解决的关键设计问题之一.以下是众核处理器系统特有的技术参数限制和负载特性:

a)互联结构发生变革.共享总线和交叉开关这种有序连接结构不再适合大规模众核.无序的、点对点的网络连接结构因具备可扩展性而被作为未来众核处理器的片上连接结构.

b)处理器核数目众多.基于点对点连接网络形成的瓦片结构(tiled)将是众核处理器结构的理想选择,而瓦片结构的众核处理器呈现出新的特征:核间通信延迟随着核数目增多急剧增大;维护一致性的硬件逻辑随着处理器核数目呈线性增长,一致性消息量剧增.

c)应用的差异性.云计算和服务器应用作为众核处理器系统的主要负载,均具有异构性和不均衡性.这种特性导致单个一致性协议无法让所有程序取得高性能.另外,在传统的目录协议中,cache 行以交叉方式分布在各个处理器节点中,这种交叉方式方便确定数据的宿主节点,但是会引入大量的 cache 一致性事务.由于在执行一致性事务之前目录协议需要获取每个存储块的共享状态,这就导致一系列的问题[19]:

a)因目录而产生的间接访问增加了生产者和消费者之间的 cache 缺失访问延迟.很多情形下,为了得到宿主节点的信息实施的目录查询操作处于 cache 访问的关键路径上.

b)自动地将共享只读类型的数据复制到很多个处理器核的本地 cache 中,减少了片上有效 cache 容量,导致 cache 缺失率上升.

c)对共享数据的写操作,或者片上目录项的替换淘汰需要将所有被共享数据的副本置为无效,增加了 cache 缺失率,增加了数据缺失处理的代价,降低了协议执行效率.根据以上分析,现存的传统多核处理器一致性协议因为性能、面积、功耗等不可扩展的原因已不适用于众核处理器,如用于 Intel Xeon[20]和 TileraTILE64[21]的一致性协议就仅适用于少数几个处理器核的情形,设计人员必须使用新的办法解决 cache 一致性问题.最近 10 年来,很多人致力于可扩展的 cache 一致性协议相关研究工作.2008 年,匹兹堡大学的 Fensch 等人[22]从软件管理的角度研究了瓦片式多核处理器的一致性策略;2009 年,麻省理工学院的 Celio 为了研究众核 cache 一致性协议,开发了一款支持 256 核的众核仿真器 Graphite[23],他们认为在众核中实现硬件 cache 一致性是可行的,但是为了得到最优的性能,软件开发者需要谨慎地编写代码,确保软件算法和目标硬件架构相匹配;同年,Intel 公司的 Dubey、Zhou 等人[24,25]认为一些面向众核处理器的并行应用算法,如 RMS,具有较少的数据共享,因此他们基于实验芯片 SCC(single-chip cloud computer)和一款 32-core 服务器初步探索了纯软件管理的一致性协议的原型系统.Kelm 等人[26]提出一种软件硬件协同控制的 Cohesion 一致性策略,Cohesion 结构在细粒度上实现一致性,当使用软件控制,Cohesion 无须片上目录,减少了很多一致性消息.

国内多家研究机构也在近几年做了不少相关研究.针对大规模多核处理器,为了减少基于目录的一致性协议中访问远程目录存储器的平均访问延时,清华大学信息科学与技术国家实验室的郭松柳、王海霞等人于 2009 年提出面向 CMP 结构的层次结构 cache 目录[27].面向片上众核处理器方面,中国科学院计算技术研究所的黄河等人[28]针对目录一致性协议存在难于实现、验证复杂和存储空间开销大等问题,提出一种由硬件结构支持、基于同步的高速缓存一致性协议.该方案不使用目录,而是通过布龙滤波器(Bloom-filter)表示一致性信息,在并行程序的同步点维护高速缓存一致性和解决数据冲突.中国科学院计算技术研究所的徐卫志等人[29]在 2010 年研究了与其 Godson-T 众核处理器一致性协议的实现密切相关的问题——同步机制,他们认为硬件支持的同步机制性能很重要,而且专用的同步机制的扩展性好;同年,基于 64 核心的 Godson-T 众核处理器;中国科学院计算技术研究所的范东睿等人为了提高对共享数据的读取效率,使用软件对共享数据实现访问控制,维护一致性[30].

参考文献:

[1] Lenoski D, Laudon J, Gharachorloo K, Gupta A, Hennessy J. The directory-based cache coherence protocol for the DASH multiprocessor. In: Proc. of the 17th Annual Int'l Symp. on Computer Architecture. Seattle, 1990. 148-159. [doi: 10.1109/ISCA.1990.134520]

[2] Gharachorloo K, Lenoski D, Laudon J, Gibbons P, Gupta A, Hennessy J. Memory consistency and event ordering in scalable shared-memory multiprocessors. In: Proc. of the 17th Annual Int'l Symp. on Computer Architecture. Seattle, 1990. 15−26. [doi: 10.1109/ISCA.1990.134503]

[3] Hennessy JL, David A. Computer Architecture: A Quantitative Approach. 5th ed., San Francisco: Morgan Kaufmann Publishers Inc., 2011.

[4] Papamarcos MS, Patel JH. A low-overhead coherence solution for multiprocessors with private cache memories. SIGARCH Computer Architecture News, 1984,12(3):348−354. [doi: 10.1145/773453.808204]

[5] Blake G, Dreslinski RG, Mudge T. A survey of multicore processors. IEEE Signal Processing Magazine, 2009,26(6):26−37. [doi: 10.1109/MSP.2009.934110]

[6] Intel Corporation. An introduction to the Intel QuickPath interconnect. Document Number 320412-001US. 2009.

[7] Maddox RA, Singh G, Safranek RJ. Weaving High Performance Multiprocessor Fabric: Architecture Insights into the Intel QuickPath Interconnect. Intel Press, 2009.

[8] Hum HHJ, Goodman JR. Forward State for Use in Cache Coherency in a Multiprocessor System. United States Patent, 6922756,2005.

[9] Martin MMK, Hill MD, Sorin DJ. Why on-chip cache coherence is here to stay. Communications of the ACM, 2012,55(7):78−89.[doi: 10.1145/2209249.2209269]

[10] Suh T, Blough DM, Lee HHS. Supporting cache coherence in heterogeneous multiprocessor systems. In: Proc. of the Conf. on Design, Automation and Test in Europe (DATE 2004), Vol. 2. Washington: IEEE Computer Society, 2004. 1150−1155. [doi: 10.1109/DATE.2004.1269047]

[11] Ros A, Cuesta B, Fernandez-Pascual R, Gomez ME, Acacio ME, Robles A, Garcia J, Duato J. Extending magny-cours cache coherence. IEEE Trans. on Computers, 2012,61(5):593−606. [doi: 10.1109/TC.2011.65]

[12] Hackenberg D, Molka D, Nagel WE. Comparing cache architectures and coherency protocols on x86-64 multicore SMP systems. In:Proc. of the 42nd Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO). New York, 2009. 413−422. [doi: 10.1145/1669112.1669165]

[13] Alisafaee M. Spatiotemporal coherence tracking. In: Proc. of the 45th Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO-45). Washington: IEEE Computer Society, 2012. 341−350. [doi: 10.1109/MICRO.2012.39]

[14] Kurian G, Khan O, Devadas S. The locality-aware adaptive cache coherence protocol. In: Proc. of the 40th Annual Int'l Symp. on Computer Architecture (ISCA). New York: ACM Press, 2013. 523−534. [doi: 10.1145/2485922.2485967]

[15] Zhao HZ, Shriraman A, Kumar S, Dwarkadas S. Protozoa: Adaptive granularity cache coherence. In: Proc. of the 40th Annual Int'l Symp. on Computer Architecture (ISCA). New York: ACM Press, 2013. 547−558. [doi: 10.1145/2485922.2485969]

[16] Luo L, Sriraman A, Fugate B, Hu S. LASER: Light, accurate sharing detection and repair. In: Proc. of the 2016 IEEE Int'l Symp.on High Performance Computer Architecture (HPCA). Barcelona, 2016. 261−273. [doi: 10.1109/HPCA.2016.7446070]

[17] Park S, Prvulovic M, Hughes CJ. PleaseTM: Enabling transaction conflict management in requester-wins hardware transactional memory. In: Proc. of the 2016 IEEE Int'l

Symp. on High Performance Computer Architecture (HPCA). Barcelona, 2016. 285−296.[doi: 10.1109/HPCA.2016.7446072]

[18] Badr M, Jerger NE. SynFull: Synthetic traffic models capturing cache coherent behaviour. In: Proc. of the 2014 ACM/IEEE 41st Int'l Symp. on Computer Architecture (ISCA). Minneapolis, 2014. 109−120. [doi: 10.1109/ISCA.2014.6853236]

[19] KHAN O,HOFFMANN H,LIS M,et al. ARCc: a case for an architecturally redundant cache-coherence architecture for large multicores[C]/ /Proc of the 29th IEEE International Conference on Computer Design. Washington DC: IEEE Computer Society,2011: 411-418.

[20] CHAIKEN D,FIELDS C,KURIHARA K,et al. Directory-basedcache coherence in large-scale multiprocessors[J]. Computer,1990,23(6) : 49-58.

[21] Zhang GW, Horn W, Sanchez D. Exploiting commutativity to reduce the cost of updates to shared data in cache-coherent systems.In: Proc. of the 48th Int'l Symp. on Microarchitecture (MICRO-48). New York: ACM Press, 2015. 13−25. [doi: 10.1145/2830772.2830774]

[22] Tilera Corporation. TILE64 processor product brief [R /OL]. (2008-2009) . http: / /www. tilera. com /sites /default /files /productbriefs /PB010_TILE64_Processor_A_v4. pdf.

[23] FENSCH C,CINTRA M. An OS-based alternative to full hardware coherence on tiled CMPs[C]/ /Proc of the 14th International Symposium on High Performance Computer Architecture. 2008: 355-366.

[24] CELIO C P. Cache coherence strategies in a many-core processor[D]. Cambridge: Massachusetts Institute of Technology,2009.

[25] DUBEY P. Recognition,mining and synthesis moves computers to the era of tera[R].[S. l.]: Intel Technology@ Corporation,2005.

[26] ZHOU Xiao-cheng,CHEN Hu,LUO Sai,et al. A case for software managed coherence in many-core processors [C]/ /Proc of the 2nd USENIX Workshop on Hot Topics in Parallelism. 2010.

[27] KELM J H,JOHNSON D R,TUOHY W,et al. Cohesion: a hybrid memory model for accelerators[C]/ /Proc of the 37th International Symposium on Computer Architecture. New York: ACM,2010: 429-440.

[28] GUO S,WANG H X,XUE Y B,et al. Hierarchical cache directory for CMP[J]. Journal of Computer Science and Technology,2010,25(2) : 246-256.

[29]黄河,刘磊,宋风龙,等. 硬件结构支持的基于同步的高速缓存一致性协议[J]. 计算机学报,2009,32(8) : 1618-1630.

[30]徐卫志,宋风龙,刘志勇,等. 众核处理器片上同步机制和评估方法研究[J]. 计算机学报,2010,33(10) : 1777-1787.

[31] 包尔固德,李伟生,范东睿,等. Godson-T 众核体系结构上的 Broadcast 性能优化[J]. 计算机研究与发展,2010,47(3) : 524-531.