计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2017.20160141
54(4):764-774, 2017

一种基于共享转发态的多级缓存一致性协议

陈继承 李一韩 赵雅倩 王恩东 史宏志 唐士斌
(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)
(chenjch@inspur.com)

A Shared-Forwarding State Based Multiple-Tier Cache Coherency Protocol

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

Abstract In CC-NUMA architecture system, in order to reduce the overhead of cache coherency maintenance, large scale CC-NUMA system usually employs multi-tier cache coherency domain method, so as to effectively alleviate the contradictions between system scalability and coherency maintenance overhead. The traditional MESI, MESIF, MOESI protocols mainly aim at the single tier coherency domain, and do not take into account the characteristic that query business accounts for dominant in the large database applications, therefore there are many problems such as high frequency of cross domain operations, low execution efficiency and so on when the protocols are used in multi-tier cache coherency domain. To address the above problems, this paper presents a multi-tier cache coherency protocol called MESI-SF based on the shared-forwarding state. The protocol creates a shared-forwarding state called Share-F, and thus there exists remote data copy with readable forwarding state in multiple coherency domains at the same time. In this way, within the same domain data copy with shared-forwarding state can directly response to read requests, and thus the protocol can effectively reduce the number of cross domain operations and enhance the system performance. Experimental simulation with SPLASH-2 test suites shows that, for the two tiers cache coherency domain system, compared with the MESI protocol, MESI-SF can reduce 23.0% visits that cross the clumps, and the average instruction execution cycle is reduced by 7.5%; compared with the MESIF protocol, MESI-SF can reduce 12.2% visits that cross the clumps, and the average cycles per instruction (CPI) is reduced by 5.95%.

Key words multi-core processes; CC-NUMA system; multi-tier coherency domain; cache coherency; MESI protocol

摘 要 在CC-NUMA架构系统中,为了减少缓存一致性维护的开销,大规模CC-NUMA系统通常采用多级缓存一致性域设计,降低平均一致性维护操作数量,从而有效缓解系统性能扩展与一致性维护开销的矛盾.传统的MESI,MESIF,MOESI协议主要是针对单级一致性域优化设计,并且没有考虑到大型数据库应用中查询(数据读访问)业务量占据主导地位的特点,故该类一致性协议在多级缓存一致性域场景下存在着跨域操作频度高、执行效率低等缺点.针对上述问题,提出了一种基于共享转发态的多

收稿日期:2016-03-11;**修回日期**:2016-06-03
基金项目:国家"八六三"高技术研究发展计划基金项目(2013AA011701)
This work was supported by the National High Technology Research and Development Program of China (863 Program) (2013AA011701).
通信作者:赵雅倩(zhaoyaqian@inspur.com)

级缓存一致性协议 MESI-SF. 该协议创建了一个共享转发态 Share-F,允许多个一致性域内同时存在远端数据副本的可读可转发状态,从而能够为同一域内同地址的读请求直接提供共享数据,有效减少了跨域操作,提升系统性能. SPLASH-2 程序集模拟结果表明,对于两级 Cache 一致性域系统,相比 MESI 协议,MESI-SF 能够减少 23.0%跨结点访问次数,指令平均执行周期数(cycles per instruction, CPI)降低 7.5%;相比 MESIF 协议,MESI-SF 能够减少 12.2%跨结点访问次数,指令平均执行周期数降低 5.95%.

关键词 多核处理器;CC-NUMA 系统;多级一致性域;缓存一致性;MESI 协议

中图法分类号 TP303

当前服务器为多处理器系统,通常采用高速缓存一致性非均匀存储访问(cache coherence non-uniform memory access, CC-NUMA)架构[1]. 在 CC-NUMA 架构系统中,缓存一致性维护开销是影响系统性能有效扩展的关键因素[2]. 为了提升系统性能,研究人员针对一致性协议展开了大量的研究工作. 在基于监听的一致性协议中,为了保证系统内所有 Cache 内容的一致性,需要在系统范围内广播一致性消息. 监听协议实现较简单、延迟较小,一般经过 2 跳即可响应请求并返回数据[3]. 由于监听协议对共享态数据的访问需要在系统内广播请求消息,容易造成网络拥塞,因此适用于 4 路以下小规模系统. MESI 协议[4]定义了 4 种数据状态,约定其他处理器对共享态数据的访问需要向数据所在内存发送请求获取,因而限制了共享请求的广播消息数量,提升了系统性能. Intel 公司进一步扩展了 MESI 协议,提出了 MESIF 协议[5],定义了一种新的共享态——转发态(F 态),使得系统中具有该状态的共享副本能够响应其他处理器的共享读请求,从而可以消除系统中的多余数据应答消息. 为了允许写状态下的数据能够直接转发共享数据给其他请求的处理器,AMD 公司在 MESI 协议的基础上引入所有态(O 态),提出了 MOESI 协议[6]. 这种方法假定程序运行过程中的回写是非必需的,因而能够减少大量回写操作消耗的存储器带宽,提升系统性能.

金融、电信等国家关键性基础设施信息系统核心业务是基于大型关系型数据库的联机事务处理类(OLTP)业务. 该类业务通常具有任务难以分解、数据关联度高、响应实时性要求高的特点[7]. 需要基于大规模 CC-NUMA 架构的高端服务器来承载该类业务. 为了提升系统的性能,通常采用具有多级缓存一致性域的 CC-NUMA 架构. 多级缓存一致性域的基本思想是分层划域[8-9],通过构建多个一致性协同芯片(coherence chip, CC)将系统划分为 2 个或多个一致性域,使得每个一致性协同芯片内部形成一个统一的逻辑 Cache 一致性域,结点(clump)间通过二级一致性协同芯片构建 Cache 一致性域. 通过这种方式构建的系统,可以减少多处理器计算机系统中一致性协同芯片的数量和结点间互连的规模,降低结点间拓扑复杂度,提高 CC-NUMA 系统的可扩展性.

然而,上述一致性协议主要是针对单级一致性域优化设计的,在当前的多级缓存一致性域体系结构下协议存在着执行效率低的问题,影响大规模 CC-NUMA 系统的性能. 如 MESI 协议应用在多级 Cache 一致性域上会导致各结点、各处理器频繁地从内存中读取共享数据,产生大量的数据交互操作,加剧网络拥塞,严重降低系统性能. MESIF 协议虽然引入了 F 态,但在多级 Cache 一致性域系统上容易造成 F 态在不同结点、不同一致性域之间来回迁移,产生颠簸效应,进而产生较大延迟,使得系统性能难以有效扩展. MOESI 协议扩展到多级 Cache 一致性域上状态复杂,同时处于 O 态的数据如果长时间存放在缓存中,一旦数据意外产生丢失没有及时写回存储设备,将造成不可预知的后果,这使得 MOESI 协议无法在大型联机事务处理业务中得到广泛应用.

同时,当前的大型数据库应用中查询业务量占据主导地位,读共享操作要远多于写独占操作. TPC 国际性能委员会在线交易测试模型的数据显示,在线交易程序的读共享和写独占的比率规定为 4.3∶1[10],在实际的应用中这一比例甚至达到 10∶1. 对于 Cache 一致性协议的性能而言,提高缓存数据之间的读共享效率显得尤为重要. 因此,结合大型数据库应用的特点,针对传统一致性协议在多级一致性域系统中应用存在的问题,本文提出了一种基于共享转发态的多级缓存一致性协议 MESI-SF. 该协议创建了一个共享转发态 Share-F,允许多个一致性域内同时存在远端数据副本的可读可转发状态,从而能够为同一域内同地址的读请求直接提供共享数据,有效减少了跨域操作,提升系统性能.

1 研究背景与动机

CC-NUMA 系统是一种典型的分布式共享内存多处理器系统. 具有两级 Cache 一致性域的 CC-NUMA 系统的一般结构如图 1 所示.

在图 1 中,系统的 n 个结点通过域间网络互连,其中每个结点内的处理器和一致性协同芯片构成了结点内 Cache 一致性域,而结点间通过一致性协同芯片互连构成了结点间 Cache 一致性域.

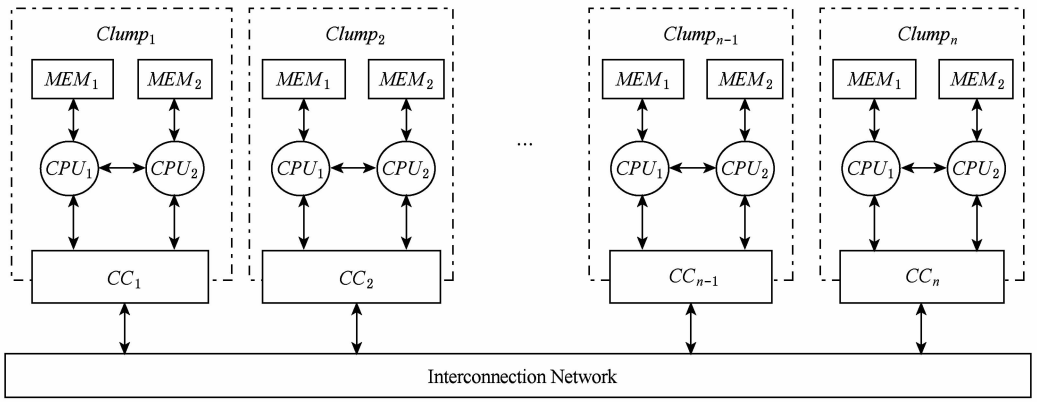


Fig. 1 The structure of multiprocessor systems with multiple clumps(each clump contains two CPUs)

图 1 多结点多处理器系统结构示意图(每结点含 2 个 CPU)

多级 Cache 一致性域系统通常采用多级 Cache 一致性协议. MESI 协议[4]是一种被广泛采用的 Cache 一致性协议,它定义了 4 种数据状态,包括:

1) M(modified)态. M 态为修改态,表明该缓存数据被修改,与内存中的数据内容不一致,为整个系统的唯一最新副本.

2) E(exclusive)态. E 态为独占态,表明当前副本为该缓存数据在系统中的唯一副本,并且未被修改,与内存中的数据内容一致.

3) S(shared)态. S 态为共享态,表明该数据在一个或多个处理器中有副本,而且副本数据未被修改,与内存中的数据内容一致.

4) I(invalid)态. I 态为无效态,表明 CPU 中该缓存数据已经失效,对其缓存行可直接覆盖替换而不需要执行缓存一致性操作.

在 MESI 协议中,当有处理器请求读共享数据时,需要通过向数据所在的内存发出请求从而才能得到该共享数据. Intel 在 MESI 协议的基础上引入了 F 转发态,提出了 MESIF 协议[5]. F 态表明该缓存数据在某一处理器中处于具有转发功能的共享状态,并且在其他处理器中可能具有一个或多个相同的但不具备转发功能的 S 态数据副本. 当其他 CPU 请求共享该数据时,需向具有 F 状态的数据副本发出请求,这一定程度上降低了共享延迟,提高了系统效率. AMD 公司在 MESI 协议的基础上引入所有态(O 态),提出了 MOESI 协议[6]. 当其他处理器请求读共享数据时,处于 O 态的缓存数据负责提供数据,这点与 MESIF 中的 F 态作用类似. 处于 O 态的缓存数据可能与存储器中对应的数据不一致,即存储器中的数据可能不是最新的,这点与 MESIF 中的 F 态不同.

在具有多级 Cache 一致性域的 CC-NUMA 系统中,上述协议中共享态数据的转发可能需要跨结点访问完成. 这里以 MESI 协议为例进行说明,结合图 1 进行说明. 假设结点 1 中 CPU_2 具有对应地址为 $addr$ 的 S 态缓存数据,$addr$ 的内存地址位于结点 2 的 MEM_1 中. 此时结点 1 中的 CPU_1 发起了读共享该数据的请求. 图 2(a)给出了这种情况下的数据请求流程图. 可以看出,由于结点 1 中 CPU_2 的共享态缓存数据无法进行转发,因而结点 1 的 CPU_1 需要跨结点通过 CC_1,CC_2 才能最终请求得到数据. 在这种情形下,由于需要跨结点访问,这一过程具有较高延迟. 观察到结点 1 中 CPU_2 的缓存数据已处于 S 共享状态,如图 2(b)所示,当结点 1 中的 CPU_1 发起同地址的数据共享读请求时,如果同结点内的 CPU_2 能够进行该数据的共享转发,那么共享请求将在结点内直接完成,而不必跨结点访问结点 2,这样将提高数据的共享效率.

上面所述问题在 MESIF,MOESI 协议中同样存在. 因而,基于上述研究动机,本文针对现有协议在多级 Cache 一致性域 CC-NUMA 中存在的跨结点访问效率较低的不足提出了一种基于共享转发态

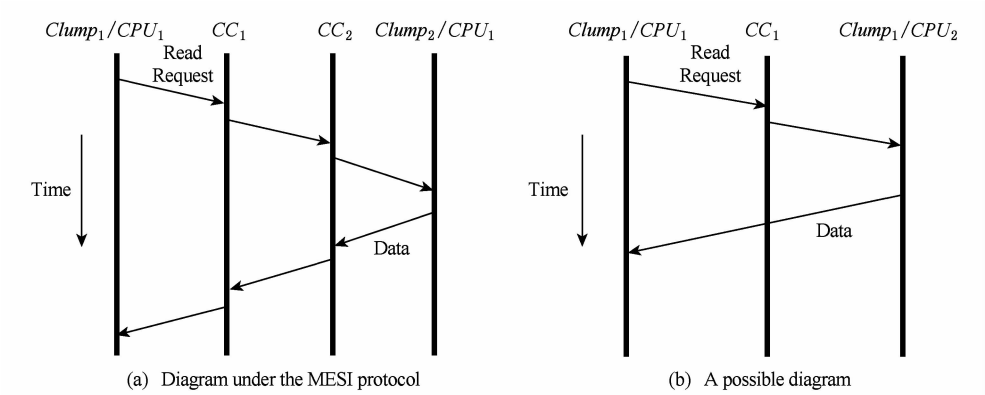


Fig. 2 Diagrams of data request process for visiting remote clumps

图 2 访问远端结点数据请求流程图

的多级缓存一致性协议,试图缓解上述存在问题,以期减少跨结点访问频度,降低数据访问延迟,提升系统性能.

2 一种基于共享转发态的多级缓存一致性协议

本文提出的多级缓存一致性协议 MESI-SF,在 MESI 协议的基础上,引入了共享转发态 Share-F. Share-F 的基本思想是通过在两级 Cache 一致性域或多级 Cache 一致性域 CC-NUMA 系统中的局部域(如结点内 Cache 一致性域)对同地址 S 状态的缓存数据在每个 Cache 一致性域内均构造一个 Share-F 状态,Share-F 状态的缓存数据可对同一域内的读请求直接提供共享数据. 本方法在不违反全局 Cache 一致性协议规则基础上可降低处理器跨结点访存频度和开销,提升 CC-NUMA 系统性能.

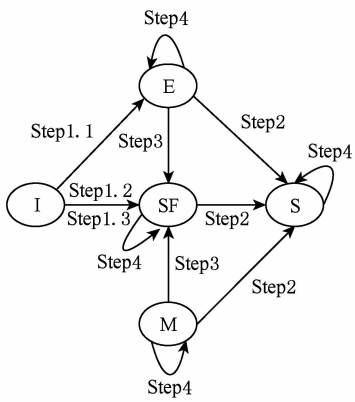
不失一般性,在本文的后续讨论中,将以两级 Cache 一致性域为例进行说明. 多级 Cache 一致性域可以看作是两级 Cache 一致性域的扩展. 本文提出的方法适用于多级 Cache 一致性域.

MESI-SF 协议包含了 MESI 协议中的 4 种状态,这 4 种状态的定义和相互之间的转换与 MESI 协议相同. 下面重点对本文提出的 Share-F 状态进行介绍. Share-F 状态适用于结点内 Cache 一致性域处于 S 状态的缓存数据. 具体地,在同一个域内,当存在同地址状态为 S 的缓存数据时,可为其中的缓存数据在域内构造唯一一个 Share-F 状态. Share-F 状态数据的意义是在结点内 Cache 一致性域中 Share-F 状态缓存数据是具有转发能力的 F 态数据副本,而在结点间 Cache 一致性域中 Share-F 状态缓存数据是不具备转发能力的 S 态数据副本.

从以上描述可知,当处理结点间 Cache 一致性事务时,Share-F 态缓存数据具有与 S 态数据相同的逻辑行为;当处理结点内 Cache 一致性事务时,Share-F 态缓存数据具有与 F 态数据相同的逻辑行为. 下面将分别从读请求和写请求 2 方面对具有 Share-F 状态的 MESI-SF 协议进行描述,最后通过示例分别对 MESI-SF,MESI,MESIF 的应用差异进行详细说明.

2.1 读共享请求

图 3 给出了 MESI-SF 协议在 Cache 一致性域内处理器读请求情况下的状态转换图. 图 3 中 SF 表示 Share-F 状态.



Step1. 1. Local read miss, and get data from memory
Step1. 2. Local read miss, and get data from other clumps
Step1. 3. Local read miss, and get data from other CPUs in the same clump
Step2. Read miss within the clump
Step3. Read miss between clumps
Step4. Local read hit

Fig. 3 Transition diagram for read request

图 3 读请求下的状态转换图

结合图 3,当结点内具有 I 状态缓存数据的处理器 CPU_1 发起对同地址缓存数据的读请求时,根据当前系统内其他处理器的状态,CPU_1 完成读请求事务后该缓存数据的状态有 3 种情况:

1) 如果请求的数据在当前其他所有处理器中

不存在副本,需要从内存中读取数据,则缓存数据在 CPU_1 中将为 E 状态;

2) 如果请求的数据由结点间其他任一处理器响应发送,则缓存数据在 CPU_1 中将为 SF 状态;

3) 如果同结点内存在同地址缓存数据状态为 SF 的处理器时,缓存数据由该处理器响应发送,请求完成后,缓存数据在 CPU_1 中将为 SF 状态.

当 CPU_1 中的缓存数据处于 M/E 其中之一的状态时,处理器 CPU_2 发起对同地址数据读请求时,根据发起读请求时 CPU_2 处理器的状态,读请求事务完成后 CPU_1 中缓存数据的状态有 2 种情况:

1) 如果 CPU_2,CPU_1 位于同一结点内,则缓存数据将变为 S 状态;

2) 如果 CPU_2,CPU_1 位于不同的结点内,则缓存数据将变为 SF 状态.

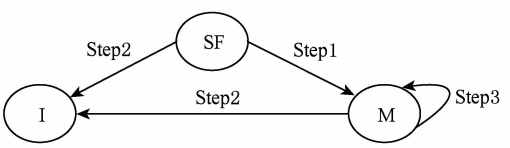
当 CPU_1 中的缓存数据处于 SF 状态时,同结点内的其他处理器发起了对同地址数据的读请求时,读请求事务完成后 CPU_1 中缓存数据将变为 S 状态.

图 3 中其他状态之间的相互转换与 MESI 协议相同,这里不再重复叙述.

由上面的分析可以看出,引入 SF 状态后,当结点内存在 SF 态缓存数据时,结点内其他处理器请求对同地址数据的读共享时,都将由 SF 态缓存数据直接响应提供数据. 这种情况下请求事务直接在结点内响应并完成,跨结点访问将减少,从而降低数据访问延迟.

2.2 写独占请求

图 4 给出了 MESI-SF 协议在 Cache 一致性域内处理器写请求情况下的部分状态转换图. 在写请求情况下的其他状态转换图可参考 MESI 协议[4].



Step1. Local write miss
Step2. Write miss within or between clumps
Step3. Local write hit

Fig. 4 Transition diagram for write request

图 4 写请求下的状态转换图

结合图 4,当结点内的处理器 CPU_1 中的缓存数据处于 SF 状态时,当有处理器 CPU_2 发起同地址数据写请求时,根据发起读请求时 CPU_2 处理器所处位置,写请求事务完成后 CPU_1 中缓存数据的状态有 2 种情况:

1) 如果 CPU_2,CPU_1 为同一处理器,即 CPU_1 发起了写请求,此时产生了写缺失,那么最终缓存数据将变为 M 状态;

2) 如果 CPU_2,CPU_1 不是同一处理器,无论 CPU_2,CPU_1 位于同一结点内还是结点间,缓存数据最终都将变为 I 状态.

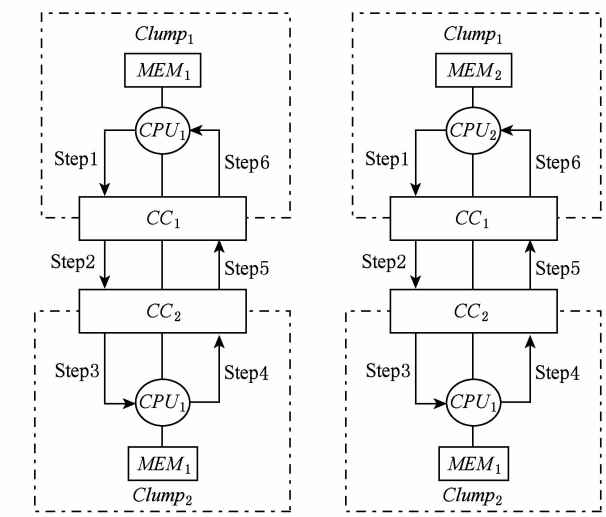
可以看出,Share-F 态在写请求下的事务处理逻辑与 S 态相同.

2.3 示例说明

本节通过示例分别说明 MESI-SF,MESI,MESIF 在数据共享方面的差异.

2.3.1 MESI-SF 与 MESI 对比

结合图 1,假设在当前状态下,结点 1 的 CPU_1 中 *addr* 地址缓存数据的状态处于 I 态,同时该缓存数据的内存地址位于结点 2 的 MEM_1 中,CC_2 中记录该缓存数据为 I 态. 图 5 和图 6 分别给出了 MESI,MESI-SF 的数据获取过程,其中图 5(a)、图 6(a)都是结点 1 的 CPU_1 请求读同地址缓存数据,图 5(b)、图 6(b)都是结点 1 的 CPU_2 请求读该缓存数据.



(a) CPU_1 of $Clump_1$ makes read request for data in MEM_1 of $Clump_2$

(b) CPU_2 of $Clump_1$ requests the same data

Fig. 5 Data retrieval under the MESI protocol

图 5 MESI 协议下数据获取过程

结合图 5(a),CC_1 结点内的处理器 CPU_1 对远端 CC_2 结点内处理器 CPU_1 处的缓存数据地址 *addr* 进行读共享访问. 在 MESI 协议下其访问过程如下:

Step1. 处理器发出访问请求操作在本地缓存未命中,向 CC_1 发出访问 *addr* 地址的内存数据请求.

Step2. CC_1 经过域间互连网络向远端 CC_2 结点发送访问请求消息.

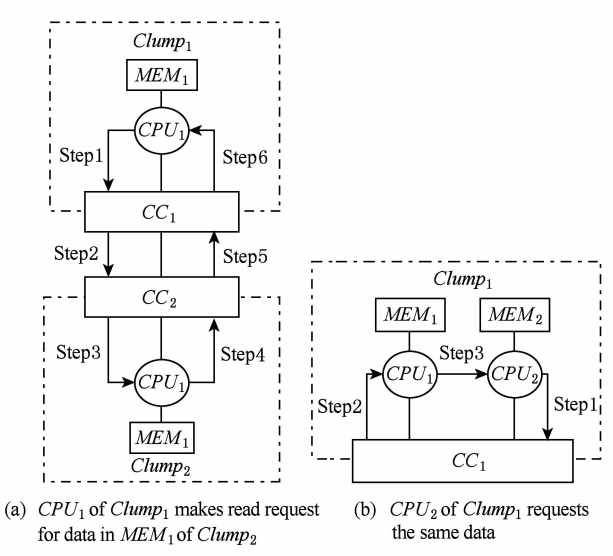


Fig. 6 Data retrieval under the MESI-SF protocol

图 6 MESI-SF 协议下数据获取过程

Step3. CC_2 查询共享信息后发现对应 $addr$ 地址的数据副本在结点内处于 I 态后,则向 CPU_1 转发数据请求.

Step4. CPU_1 接收到数据请由求后,从内存 MEM_1 读取数据,并将数据返回给 CC_2.

Step5. CC_2 收到数据后,将 CPU_1 对应 $addr$ 地址的数据副本状态由 I 态改为 S 态,同时将数据信息转发给 CC_1.

Step6. CC_1 接收到 CC_2 转发的数据信息后,将该数据信息发送给 CPU_1,同时将对应 $addr$ 地址的数据副本状态由 I 改为 S 态.

完成图 5(a)中的事务后,紧接着,图 5(b)中 CC_1 结点内的另一处理器 CPU_2 同样要访问远端 CC_2 结点内处理器 CPU_1 中 $addr$ 地址的缓存数据,其访问过程与图 5(a)类似. 可以看到,虽然此刻同一结点内的 CPU_1 具有 S 态的共享数据,但 CPU_2 仍然需要跨结点并从内存中读数来获取共享数据.

接着,图 6(b)中,CC_1 结点内的处理器 CPU_2 同样对 $addr$ 地址的缓存数据进行访问. 其访问过程如下:

Step1. 处理器发出访问请求操作在本地缓存未命中,向 CC_1 发出访问 $addr$ 地址的内存数据请求.

Step2. CC_1 查询共享信息后,发现 CPU_1 中对应 $addr$ 地址的缓存数据状态为 Share-F 态后,向 CPU_1 转发数据请求信息.

Step3. CPU_1 接收到数据请求后,向 CPU_2 发送对应 $addr$ 地址的缓存数据.

图 6(b)中,由于 CPU_1 中该缓存数据状态为 Share-F,因而 CPU_2 请求的数据将由 CPU_1 直接提供.

通过对比图 5(b)和图 6(b),容易看出在相同场景下,当结点内有共享态数据存在时,结点内其他处理器发起读共享该数据,在 MESI 协议下需要跨结点访问内存才能获取到数据,具有较大的延迟;而在 MESI-SF 协议下共享数据获取在同一结点内完成,无需跨结点访问内存,具有较小的延迟.

2.3.2 MESI-SF 与 MESIF 对比

本节通过示例详细说明 MESI-SF 与 MESIF 协议在数据共享方面的差异. 结合图 1,假设结点 1 中的 CPU_1 和结点 2 中的 CPU_1 先后对位于结点 2 中的同地址数据进行读访问. 当 2 次读访问完成后,在 MESI-SF 协议下,结点 1 中存在 SF 态的该缓存数据;而在 MESIF 协议下,结点 1 中存在 S 态的该缓存数据,结点 2 中存在 F 态的该缓存数据. 当结点 1 中的 CPU_2 接着发起对该缓存数据的读访问时,图 7 列出了 2 种协议在这种状态下的数据获取过程. 通过对比图 7(a)和图 7(b),不难发现,在相同场景下,MESI-SF 协议只需在结点内完成数据获取过程,而 MESIF 需要跨结点访问 F 态数据才能完成数据获取过程,产生较大访问延迟. 特别地,当结点之间连续频繁交替访问同地址数据时,MESIF 协议中的 F 态数据会频繁在结点之间迁移,具有较大的延迟;而 MESI-SF 在这种情形下,由于结点内在访问过同地址数据后,结点内已存在 SF 态数据,因而接下来对同地址数据的访问只在结点内完成,具有较小的延迟.

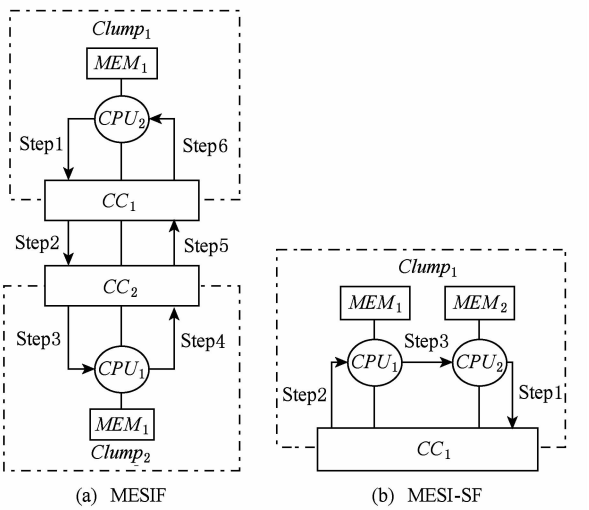


Fig. 7 Data retrieval for two protocols

图 7 2 种协议数据获取过程

3 性能评估

3.1 实验平台

本文在 gem5[11]模拟器上实现了 MESI,MESIF,MESI-SF 协议,并构建了多级 Cache 一致性域系统.在实验中,gem5 的参数配置如表 1 所示.为了验证本文提出方法的有效性,本文采用 SPLASH-2 测试程序集[12],它被广泛地应用于验证 Cache 一致性协议的正确性及有效性[13-14].同时,SPLASH-2 测试程序集中平均读写比达到了 3∶1,其中 CHOLESKY 的读写比甚至达到了 10.4∶1.因而,SPLASH-2 程序集表现出来的高读写比可以很好地用来模拟大型关系型数据库中较频繁的读共享这一特性.表 2 列出了测试程序及其数据输入情况.

Table 1 Configuration of gem5

表 1 gem5 模拟器参数配置

Modules	Configurations
Processor	64 cores
Clumps	4,8,16
CPU Frequency/GHz	1.5,2
Memory Size/GB	1
Cache Hierarchy	Inclusive
Cache Size/B	64
Private L1 I/D Cache	4 ways, 32(I)+64(D)KB, write back
Shared L2 Cache	8 ways, 256KB/Core, write back
Network	2D mesh
Ratio of Latency	1∶1∶1

Table 2 Data Input of SPLASH-2 Programs

表 2 SPLASH-2 测试程序及其数据输入

Programs	Data Input
CHOLESKY	tk18.O
FFT	256×1 024 Points
RADIX	1 048 576 keys, 1 024 radix
LU	1 024×1 024 matrix,16×16 blocks
NLU	1 024×1 024 matrix,16×16 blocks
OCEAN	258×258
BARNES	16 384 particles, time-step is 0.025

3.2 评估结果

在实验评估过程中,我们首先与 MESI,MESIF 协议方法分别在 2 GHz 主频、8 结点规模系统下进行了实验比较;接着,我们评估了不同主频、读写比和结点数对协议性能的影响.

3.2.1 指令平均执行周期数(CPI)

图 8 给出了 MESI,MESIF,MESI-SF 协议在各个程序测试集上的指令平均执行周期数(CPI)比较数据.图 8 的所有实验数据均经过相对于 MESI 协议方案的归一化处理.从图 8 可以看出,与 MESI 协议相比,MESI-SF 在所有测试程序上的相对指令执行性能都小于 1,指令平均执行周期数降低 7.5%;与 MESIF 协议相比,MESI-SF 在所有测试程序上的相对指令执行周期数更小,指令平均执行周期数降低 5.95%.这说明了程序在 MESI-SF 协议上表现出了更少的平均执行时间.图 9 进一步给出了 3 个协议在各个程序测试集上的平均读缺失延迟比较情况下.通过分别对比相同程序在平均指令执行周期数和平均读缺失延迟的表现可以看出,与 MESI 和 MESIF 协议相比,采用 MESI-SF 协议,在所有 CPI 出现减少的程序上平均读缺失延迟也减少了.这是因为相比其他 2 个协议,MESI-SF 在结点内引入了 Share-F 状态,原先在 MESI 和 MESIF 协议中需要跨结点访问的请求,在 MESI-SF 中可能只需在结点内通过 Share-F 状态的数据就能获取到共享数据,这降低了访问延迟,因而 CPI 出现了减少.

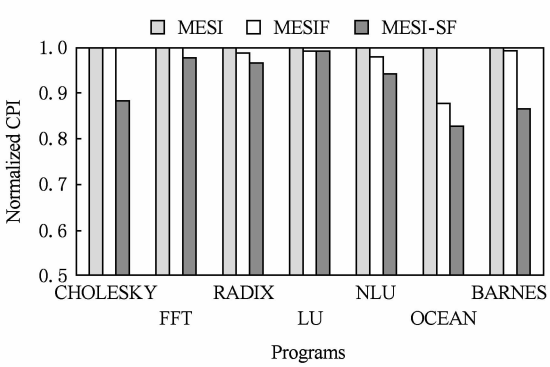


Fig. 8 Comparison of normalized CPI

图 8 归一化 CPI 比较

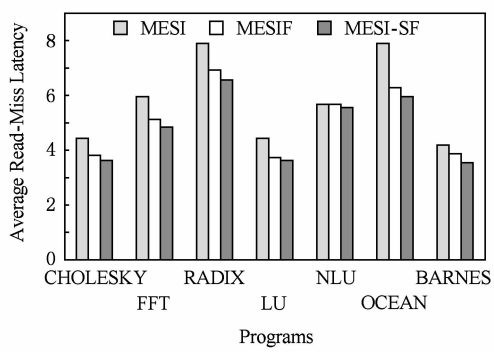


Fig. 9 Comparison of average read-miss latency

图 9 程序的平均读缺失延迟比较

3.2.2 结点内读缺失率

图 10 给出了 3 个协议在各个程序测试集上的结点内读缺失率比较情况.从图 10 可以看出,MESI 协议在所有程序测试集上表现出较高的结点内读缺失率,平均读缺失率为 98.9%;MESIF 结点内读缺失率为 86.7%;MESI-SF 在所有程序测试集上都表现出了较低的结点内读缺失率,与 MESI 和 MESIF 相比结点内读缺失率平均降低 23%,12.2%.3 个协议在程序集上呈现出了不同的结点内读缺失率,这是由于当结点内有其他处理器请求读共享内存地址位于其他结点内的数据时,MESI 协议不管结点内存不存在共享态数据,总是需要跨结点访问来获取数据,因而读缺失率较高;当结点内存在同地址 F 态可转发的数据时,MEISF 协议对这类读请求可在结点内由 F 态数据转发进行共享;MESI-SF 协议当结点内存在 1 个或多个拥有共享态数据的处理器时,共享请求可由其中一个拥有共享转发态数据的响应转发完成,从而降低了跨结点访问的请求.因而从以上分析可以看出,与 MESI,MESIF 相比,MESI-SF 能够有效地减少跨结点读访问次数,降低了结点内读缺失率,从而提升数据共享效率.

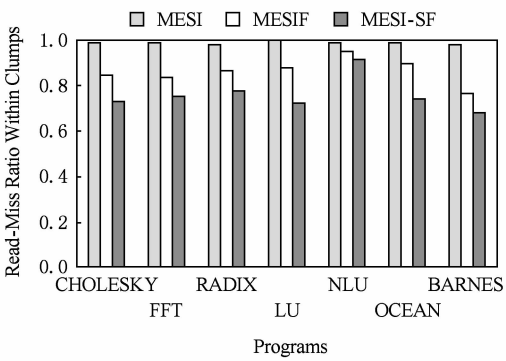


Fig. 10 Comparison of read-miss ratio within clumps

图 10 结点内读缺失率比较

3.2.3 CPU 主频对性能的影响

本节研究不同 CPU 主频对 MESI,MESI-SF 性能的影响.图 11 列出了不同主频下的归一化指令执行性能比较数据.图 11 的所有实验数据均经过相对于 MESI 协议方案在 2GHz 主频下性能的归一化处理.从图 11 可以看出,在相同 CPU 主频下,在 MESI-SF 协议上运行的程序的归一化指令执行性能总是要高于 MESI,也就是说 MESI-SF 的性能效率总是要优于 MESI.进一步观察可以看到,MESI-SF 的指令执行性能曲线总是位于 MESI 的相应曲线上方.因而,当 2 个协议的运行性能相同时,即拥有相等的指令执行性能时,MESI-SF 的 CPU 主频总是要低于 MESI 的 CPU 主频.这里以 2 GHz CPU 主频下的 MESI 协议性能为例.MESI-SF 达到同等性能只需要运行在 1.82 GHz 左右.这说明了基于 MESI-SF 协议的系统在 1.82 GHz 的 CPU 主频下达到了传统 MESI 协议方法在 2GHz 主频的运行性能,提升了系统能效.

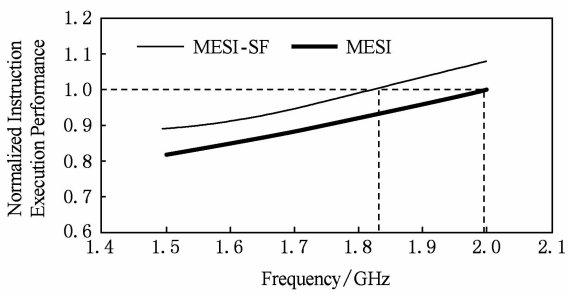


Fig. 11 Comparison of instruction execution performance for different frequencies

图 11 不同主频下的指令执行性能比较

3.2.4 读写比对 MESI-SF 性能的影响

MESI-SF 针对缓存数据的读请求进行了优化,本节进一步研究在不同程序的读写比对 MESI-SF 性能的影响.在实验中,我们收集了各个程序集运行过程中由处理器内核发起的读请求和写请求的数量,并据此得出读写比.图 12 列出了不同程序的读写比例.结合图 8 可以看出,具有较高读写比的程序在 MESI-SF 协议下其性能往往有较高程度的提升.如 CHOLESKY 和 BARNES,其读写比分别达到 10.4∶1和 4.2∶1,性能提升程度在所有程序中是较为显著的;而读写比较低的程序,如 FFT,LU,NLU,其性能提升不显著.结合图 10,可以看到,虽然 RADIX 具有较高的读写比,但其在 MESI-SF 协议下具有较高的结点内读缺失率,也就是具有较低的结点内读命中率,在这种情况下有较多的共享数据的读请求仍然需要跨结点访问完成,这说明结点内

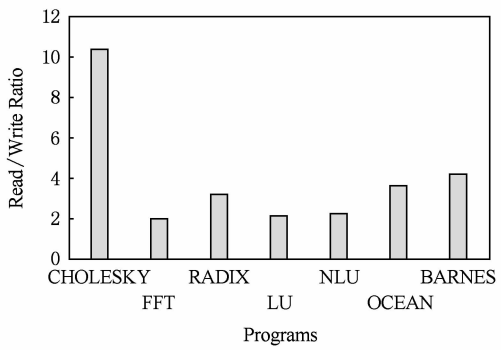


Fig. 12 Read/write ratio for different programs

图 12 不同程序的读写比

的缓存数据由于状态发生变化导致不常拥有 Share-F 态,在这种情形下本文提出的优化方法性能提升并不显著.

3.2.5 结点数对 MESI-SF 性能的影响

本节进一步研究不同结点数对 MESI-SF 性能的影响.在实验过程中统一采用 64 个处理器,分别研究在 4 结点、8 结点和 16 结点情形下 MESI-SF 性能的变化情况.图 13～图 15 分别给出了相应的实验对比结果.

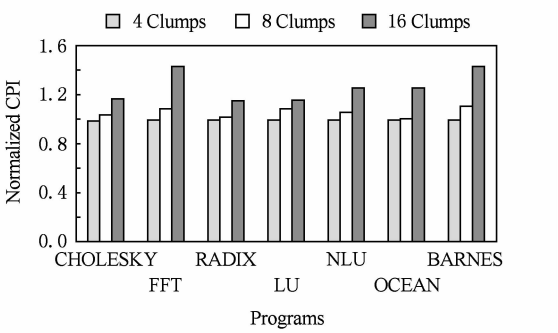


Fig. 13 Impact of the number of clumps on CPI

图 13 结点数量对 CPI 的影响

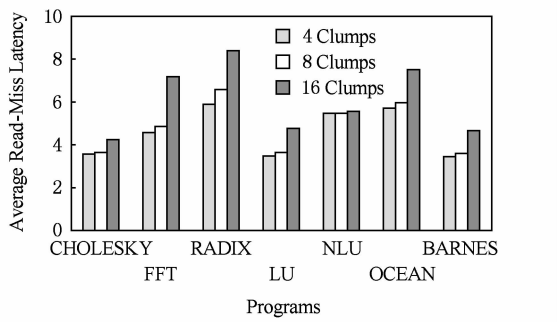


Fig. 14 Impact of the number of clumps on read-miss latency

图 14 结点数量对程序平均读缺失延迟的影响

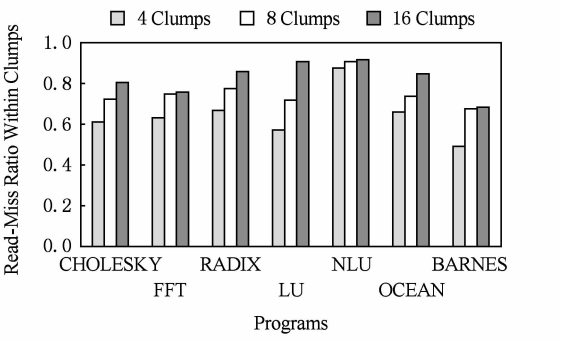


Fig. 15 Impact of the number of clumps on read-miss ratio within clumps

图 15 结点数量对结点内读缺失率的影响

结合图 13 和图 14 可以看出,随着结点数量的增加,结点内 CPU 数量减少,各个程序测试集的平均执行周期数均出现了不同程度的增加,平均读缺失延迟相应呈现增加的趋势.以 FFT 程序为例说明.当结点数量为 4 个,归一化 CPI 为 1,平均延迟为 4.89 cycle;当结点数量增加到 16 个,归一化 CPI 增加到 1.43,平均延迟增加到 7.21 cycle.这是因为,随着结点数量的增加,结点内 CPU 数量减少,那么数据被结点内 CPU 访问的概率就变小,在此基础上,结点内存在 Share-F 态数据的概率就减小,从而导致跨结点访问增加,因而 CPI 和平均延迟均呈现出了增加趋势.

另外,从图 15 可以看出,随着结点数量的增加,各个程序的结点内读缺失率呈现出增加的趋势.这也解释了上述实验中 CPI 随着结点数量的增加而增大的现象.

4 讨 论

本节对 MESI-SF 协议可能存在的问题进行讨论.

在 MESI-SF 协议中,同一结点内存在 Share-F 态的缓存数据时,其他处理器请求该缓存数据时并不一定总是会由该 Share-F 态的缓存数据提供转发数据.这是因为 Share-F 态会因替换(replacement)操作转换成 I 态,此时由于静默降级[15](silent replacement)策略而不通知一致性协同芯片.在这种情况下,同结点内读请求仍需要跨结点访问来完成.结合图 16 进行说明.结点 1 的 CPU_1 具有对应地址为 $addr_2$ 的 Share-F 态缓存数据,CPU_2 对应 $addr_2$ 的缓存数据为 I 态.同时该缓存数据的内存地址 $addr_2$ 位于结点 2 的 MEM_2 中.由于结点 1 的 CPU_1 中 $addr_2$ 地址缓存数据产生了替换(replacement)操作,其状态变为 I 态.而此时 CC_1 处仍然记录 CPU_1 具有 Share-F 态.在这种情况下,结点 1 的 CPU_2 对地址 $addr_2$ 处的数据发起了读请求访问,此时数据请求需要通过结点 2 的 CPU_2 来完成,当访问请求事务完成后,CPU_2 中 $addr_2$ 将具有 Share-F 态.具体流程如下:

Step1. CPU_2 发出访问请求操作在本地缓存未命中,向 CC_1 发出访问 $addr_2$ 地址的内存数据请求.

Step2. CC_1 查询共享信息后发现对应 $addr_2$ 地址的数据副本在本结点内 CPU_1 处为 Share-F 态,向 CPU_1 转发数据请求.

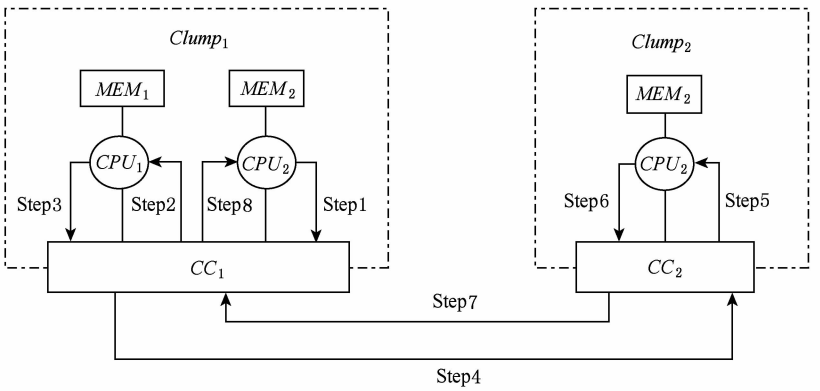


Fig. 16 Data retrieval under the condition of invalid states

图 16 无效状态下的数据获取过程

Step3. CPU_1 接收到数据请求后,发现对应 $addr_2$ 地址的缓存数据状态为 I 态,向 CC_1 发送请求无效消息.

Step4. CC_1 收到请求无效消息后,通过域间互连网络向 CC_2 发送数据请求消息.

Step5. CC_2 收到请求后发现缓存数据处于共享态,向 CPU_2 转发数据请求.

Step6. CPU_2 接收到数据请求后,从内存中读取数据,并将数据返回给 CC_2.

Step7. CC_2 收到数据后,将数据信息转发给 CC_1.

Step8. CC_1 接收到 CC_2 转发的数据信息后,将该数据信息发送给 CPU_2,同时将对应 $addr_2$ 地址的数据副本状态由 I 改为 Share-F 态.

为了避免此种情形的发生,一种可行的做法是当结点内的 Share-F 态静默降级为 I 态时,向结点一致性协同芯片发送无效(invalid)信息,然而这可能会加剧网络拥塞.我们将在以后的工作中进一步研究这种方法对性能的影响.

在目录存储开销方面,与 MESI,MESIF 相比,MESI-SF 引入了新的状态,由于在目录项的状态位上 SF 依然能够用来表示结点内有共享数据存在,因而状态位仍然可以采用 2 位编码.但是在目录项中为了指定哪一个 CPU 中数据处于 SF 状态,需要增加 lb$n$ 位来标记(其中 n 为结点内 CPU 数量),这在一定程度上会增加目录的存储开销.我们将在以后的工作中研究降低目录存储开销的方法.

5 结束语

在 CC-NUMA 架构系统中,缓存一致性维护开销是影响系统性能有效扩展的关键因素.为了减少缓存一致性维护开销,CC-NUMA 架构通常采用多级缓存一致性域.传统的 MESI,MOESI,MESIF 协议主要是针对单级一致性域优化设计,在多级缓存一致性域场景下存在着跨域操作频度高、执行效率低等缺点.针对该类一致性协议在多级一致性域系统中应用存在的问题,结合大型数据库应用中查询业务量占据主导地位的特点,本文提出了一种基于共享转发态的多级缓存一致性协议 MESI-SF.该协议创建了一个共享转发态 Share-F,允许多个一致性域内同时存在远端数据副本的可读可转发状态,从而能够为同一域内同地址的读请求直接提供共享数据,有效减少了跨域操作,提升系统性能.SPLASH-2 程序集模拟结果表明,对于两级 Cache 一致性域系统,相比 MESI 协议,MESI-SF 能够减少 23%跨结点访问次数,指令平均执行周期数降低 7.5%;相比 MESIF 协议,MESI-SF 能够减少 12.2%跨结点访问次数,指令平均执行周期数降低 5.95%.进一步降频实验表明,基于 MESI-SF 协议的系统在 1.82 GHz 的主频下达到了传统 MESI 协议方法在 2 GHz 主频的运行性能,提升了系统能效.

参 考 文 献

[1] Laudon J, Daniel L. The SGI origin: A CC-NUMA highly scalable server[J]. ACM SIGARCH Computer Architecture News, 1997, 25(2): 241-251

[2] Pang Zhengbin, Li Qiong, Li Yongjin, et al. Data coherence maintenance of distributed shared I/O in CC-NUMA system [J]. Journal of Computer Research and Development, 2007, 44(Suppl 1): 226-232 (in Chinese)
(庞征斌, 李琼, 李永进, 等. CC-NUMA 系统分布共享 I/O 的数据一致性维护[J]. 计算机研究与发展, 2007, 44(增刊 1): 226-232)

[3] Ravishanicar C, James R. Cache implementation for multiple microprocessors [C] //Proc of COMCON83. New York: ACM, 1983: 346-350

[4] Papamarcos M S, Patel J H. A low-overhead coherence solution for multiprocessors with private cache memories [J]. ACM SIGARCH Computer Architecture News, 1984, 12 (3): 348-354

[5] Kanter D. The common system interface: Intel's future interconnect [EB/OL]. [2016-01-21]. http://www.realworldtech.com/page.cfm? ArticleID=RWT082807020032

[6] Advanced Micro Devices. AMD64 Architecture Programmer's Manual: V2: System Programming [EB/OL]. [2016-01-17]. http://support.amd.com/us/Embedded_TechDocs/24593.pdf

[7] Wang Endong, Hu Leijun, Zhang Dong, et al. Design and Implementation of High-End Fault-Tolerant Computer [M]. Beijing: Tsinghua University Press, 2015 (in Chinese)
(王恩东, 胡雷钧, 张东, 等. 高端容错计算机设计与实现 [M]. 北京: 清华大学出版社, 2015)

[8] Wang Endong, Chen Jicheng, Hu Leijun, et al. Method of constructing share-f state in local domain of multi-level cache coherency domain system: USA, US20150058570 [P]. 2015-02-26

[9] Gostin, G, Jean C, Kirby C. The architecture of the HP superdome shared-memory multiprocessor [C] //Proc of the 19th Annual Int Conf on Supercomputing. New York: ACM, 2005: 239-245

[10] Chen S, Ailamaki A, Athanassoulis M, et al. TPC-E vs TPC-C: Characterizing the new TPC-E benchmark via an I/O comparison study [J]. ACM SIGMOD Record, 2011, 3(39): 5-10

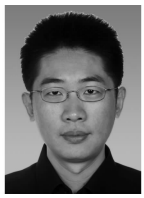
[11] Binkert N, Beckmann B, Black G, et al. The gem5 simulator [J]. ACM SIGARCH Computer Architecture News, 2011, 39(2): 1-7

[12] Woo S C, Ohara M, Torrie E, et al. The SPLASH-2 programs: Characterization and methodological considerations [J]. ACM SIGARCH Computer Architecture News, 1995, 23(2): 24-36

[13] Beckmann B M, Wood D. Managing wire delay in large chip-multiprocessor caches [C] //Proc of the 37th Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2004: 319-330

[14] Zhang Jun, Tian Ze, Mei Kuizhi, et al. Node predicting based direct cache coherence protocol for chip multi-processor [J]. Chinese Journal of Computers, 2014, 37(3): 700-720 (in Chinese)
(张骏, 田泽, 梅魁志, 等. 基于结点预测的直接 Cache 一致性协议[J]. 计算机学报, 2014, 37(3): 700-720)

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

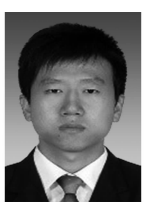


Chen Jicheng, born in 1976. Received his PhD degree from Zhejiang University. Associate professor. Member of CCF. His main research interests include computer architecture, cache coherence and integrated circuit design.

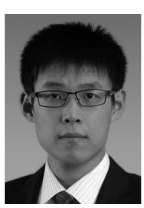
Li Yihan, born in 1985. Received his PhD degree from Beihang University. His main research interests include computer architecture and software debugging.

Zhao Yaqian, born in 1981. Received her PhD degree from Beihang University. Her main research interests include computer architecture and machine learning.

Wang Endong, born in 1966. Received his MSc degree from Tsinghua University. Professor. Fellow member of CCF. His main research interests include computer architecture, hybrid storage system and interconnect protocols for scalable system (wangendong@inspur.com).



Shi Hongzhi, born in 1988. Received his BEn degree from Xidian University. His main research interests include computer architecture and cache coherence (shihzh@inspur.com).



Tang Shibin, born in 1986. Received his PhD degree from the Institute of Computing Technology, Chinese Academy of Sciences. His main research interests include computer architecture, cache coherence and on-chip memory system (tangsb@inspur.com).