计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20170898
56(4):719-729, 2019

一种基于自更新的简单高效 Cache 一致性协议

何锡明 马 胜 黄立波 陈 微 王志英
(国防科技大学计算机学院 长沙 410073)
(heximing15@nudt.edu.cn)

A Simple and Efficient Cache Coherence Protocol Based on Self-Updating

He Ximing, Ma Sheng, Huang Libo, Chen Wei, and Wang Zhiying
(*College of Computer, National University of Defense Technology, Changsha* 410073)

Abstract As the number of cores in a chip multiprocessor increases, cache coherence protocols have become a performance bottleneck of the share-memory system. The overhead and complexity of current cache coherence protocols seriously restrict the development of the share-memory system. Specifically, directory protocols need high storage overhead to keep track of sharer list and snooping protocols consume significant network bandwidth to broadcast messages. Some coherence protocols, such as MESI (modified exclusive shared or invalid) protocol, are extremely complex and have numerous transient states and data race. This paper implements a simple and efficient cache coherence protocol named VISU (valid/invalid states based on self-updating) for data-race-free programs. VISU is based on a self-updating mechanism and only includes two stable states (valid and invalid). Furthermore, the VISU protocol eliminates the directory and indirection transactions and reduces significant overheads. First, we propose self-updating shared blocks at synchronization points for correction with the data-race-free guarantee of parallel programming. Second, taking advantage of techniques that dynamically classify private data (only accessed by one processor) and shared data, we propose write-back for private data and write-through for shared data. For private data, a simple write-back policy can reduce the unnecessary on-chip network traffic. In L1 cache, a write-through policy for shared data which can keep the newest shared data in LLC, would obviate almost all coherence states. Our approach implements a truly cost-less two-state coherence protocol. The VISU protocol does not require directory or indirect transfer and is easier to verify while at the same time obtains similar even better performance of MESI directory protocol.

Key words shared memory; chip multiprocessors; cache coherence protocol; self-updating; VISU protocol

摘 要 随着片上多处理器系统核数的增加,当前一致性协议上存在的许多问题使共享存储系统复杂而低效.目前一些一致性协议极其复杂,例如 MESI(modified exclusive shared or invalid)协议,存在众多的中间状态和竞争.并且这些协议还会导致额外失效通信,以及大量记录共享信息的目录存储开销(目录协议)或广播消息的网络开销(监听协议).对数据无竞争的程序实现了一种简单高效一致性协议 VISU(valid/invalid states based on self-updating),这种协议基于自更新操作(self-updating)、只包含

收稿日期:2017-11-28;**修回日期**:2018-04-17
基金项目:国家自然科学基金项目(61572508,61672526,61472435,61472432,61202121);国防科技大学科研计划项目(ZK-03-06)
This work was supported by the National Natural Science Foundation of China (61572508, 61672526, 61472435, 61472432, 61202121) and the Scientific Research Project of National University of Defense Technology (ZK-03-06).
通信作者:马胜(masheng@nudt.edu.cn)

2 个稳定状态(valid/invalid). 所设计的两状态 VISU 协议消除了目录和间接事务. 首先基于并行编程的数据无竞争(data race free, DRF)模型,采用在同步点进行自更新共享数据来保证正确性. 其次利用动态识别私有和共享数据的技术,提出了对私有数据进行写回、对共享数据进行写直达的方案. 对于私有数据,简单的写回策略能够简化不必要的片上通信. 在 L1 cache 中,对于共享数据的写直达方式能确保 LLC(last level cache)中数据最新从而消除了几乎所有的一致性状态. 实现的 VISU 协议开销低、不需要目录、没有间接传输和众多的一致性状态,且更加容易验证,同时获得了与 MESI 目录协议几乎相当甚至更优的性能.

关键词 共享存储;片上多处理器;cache 一致性协议;自更新;VISU 协议

中图法分类号 TP303

共享存储是当前使用最为广泛的并行编程模型,然而复杂的一致性协议使设计高效、低功耗、可扩展的多核共享存储系统变得十分困难. 为了满足一致性的定义,一致性协议必须对写操作立即响应,失效其他核 cache 备份. 这就要求目录协议实时地记录 cache 的备份并发送间接消息,或者要求监听协议广播消息去失效备份. 除此之外,当前的协议还会增加各种稳定状态(独占 E 或者拥有 O)来提高性能,增加众多中间状态以处理各种情况下出现的竞争. 例如 GEM5[1] 中实现的 MESI(modified exclusive shared or invalid)目录协议在 L1 中就有 15 个状态. 一直以来,大多数的共享存储多处理器实现 cache 一致性都是基于目录协议. 在国际上,针对 cache 一致性协议的存储开销和验证开销的优化设计层出不穷[2-7]. 而在国内,研究者们在一致性协议功能扩展和性能优化等方面取得了一系列的成果[8-10].

而实际上,在多核程序运行过程中,大部分的数据都只被一个处理器访问(甚至在程序并行阶段),我们称之为私有数据. 私有数据仅由一个处理器访问,不存在 cache 一致性的问题,不需要一致性的维护. 这种数据特性被越来越多的研究者关注,有的研究者利用这种共享特性过滤监听协议中多余的消息来降低监听协议的网络开销[11],有的研究者利用这种共享特性对目录协议的存储开销进行优化,以提高目录的利用率[2,5].

2011 年 Cuesta 等人[2]利用数据划分的方法对协议进行优化. 他们通过测试并行测试集 SPLASH-2 (Stanford parallel applications for shared memory)中的 8 个程序、scientific benchmarks 中的 2 个程序、ALPBench(all levels of parallelism benchmark for multimedia)中的 4 个程序、PARSEC(Princeton application repository for shared-memory computers)中的 4 个程序,发现平均有 75% 的数据块(64 B)是私有的. 他们在目录中忽略这些私有数据块的信息,有效地提高了目录的利用率. 实验结果表明:在相同的目录大小情况下,他们的方案获得 15% 的性能提高. 然而他们这种针对于目录利用率的优化并没有真正地解决协议中的目录开销、大量的一致性状态和间接访问等问题. 2012 年 Ros 等人[12]利用划分私有和共享数据的方法,进行动态的写策略切换. 这种写策略对私有数据进行写回,对共享数据进行写直达,简化了一致性状态. 利用并行程序的数据无竞争(data race free, DRF)模型[13],在同步点进行共享数据自失效的方法完全避免了目录的存储开销. 相对于目录协议,他们的设计平均减少了 14.2% 的系统能耗. 但是写直达方式以及同步点自失效的方法大大增加了系统 cache 的缺失率,影响系统性能.

针对上述问题,本文主要研究共享存储一致性协议的简化. 在满足 DRF 的模型下,设计了一种基于自更新操作(self-updating)只包含 2 个稳定状态(valid/invalid)的简单高效一致性协议,简称 VISU (valid/invalid states based on self-updating)协议. 协议的简化主要体现在 2 个部分:

1) 完全消除目录和间接事务. 在目录协议中,目录的主要功能是记录每个共享备份的位置以便获取最新的数据和失效过期的备份(间接事务). VISU 协议利用数据的共享特性,进行私有共享数据块的动态划分. 在共享数据方面,VISU 协议采用了写直达和自更新操作,LLC(last level cache)始终保持着最新的共享数据,同时自更新操作及时地更新过期的备份. 在私有数据方面,VISU 协议采用写回方式,在协议中不需要目录记录信息,不进行一致性的维护(类似于单核访问). 因此整个协议消除了目录和间接事务.

2) 简化成 2 个稳定状态. 通过 TLB 表项提供的页面粒度的私有共享信息,VISU 协议在 cache

line 中只需要 2 个稳定的状态(valid/invalid)就能正确地满足一致性,同时获得了与 2 种稳定状态的 MESI 协议相当甚至更优的性能.

1 VISU 一致性协议正确性论证

为了论证 VISU 协议的正确性,首先介绍关于一致性协议需要满足的 2 个条件,然后具体描述 VISU 协议,最后结合定义与描述论证 VISU 协议的正确性.

1.1 一致性的定义

Sorin 等人[13]定义一致性:1)同一时刻满足对于同一存储单元只能一个写或者多个读(single-write multiple-read, SWMR);2)在每个时期开始阶段存储单元的值都必须和最新一次的读写时期结束阶段的值相同(data value invariant),本文称之为数据最新原则.

1.2 VISU 协议的描述

本文设计一种基于自更新操作只包含 2 个稳定状的简单高效 VISU 协议. 协议可分成私有和共享 2 个独立的部分. 首先我们介绍数据无竞争的编程模型,然后介绍 VISU 协议中私有数据和共享数据的协议. 为了直观地阐述协议,假设系统为 2 级 cache,L1 cache 和 LLC(last level cache).

数据无竞争的编程模型保证在程序中无数据竞争,即当 2 个不同的线程同时访问相同的存储单元且至少有一个是写操作时,必然存在同步语句把它们分开. 数据竞争经常导致程序的错误. DRF 编程模型是 C++和 Java 高级编程语言存储模型的基础,绝大部分并行程序都满足 DRF 编程模型. 因此 VISU 协议利用了程序的 DRF 特性进行一致性的设计.

VISU 协议在同步点进行自更新操作以简化间接事务,并利用数据的共享特性,把协议分成私有和共享 2 个独立的部分:

1) 私有数据协议. 私有数据协议类似于单核处理器的数据访问,消除了私有数据冗余的一致性维护. L1 cache 采用了写回的方式.

2) 共享数据协议. 与简单的私有数据协议相比,共享数据协议在 L1 cache 中采用写直达的方式代替写回方式. 同时,共享数据在同步点进行自更新操作.

① VISU 协议针对私有数据的处理. VISU 协议对私有数据处理类似于单核处理器的处理,采用 cache 写回策略. 图 1 和图 2 展示了 2 级 cache 的 VISU 协议. 图 1 展示了协议中的主要事务读(read, Rd)、写(write, Wrt)、写回(write-back, WB). 图 2 为 VISU 的状态转换图,2 种稳定状态(V,I)和 2 种中间状态(VI, IV),其中图 2(a)为 L1 cache 控制器的状态转换图,图 2(b)为 LLC 控制器的状态转换图.

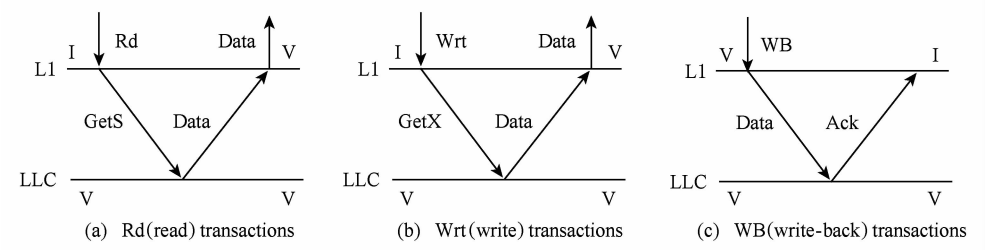

Fig. 1 VISU: Rd, Wrt and WB transactions of private data

图 1 VISU 私有数据读(Rd)、写(Wrt)、写回(WB)事务

当发生读(Rd)缺失时,如图 1(a),L1 cache 发送 GetS 到 LLC,然后获取 Data 数据;当发生写(Wrt)缺失时,如图 1(b)所示,L1 发送 GetX 到 LLC 中,LLC 返回 Data 数据. 由于数据为私有数据,其他核不存在数据的备份,所以读缺失和写缺失不需要向其他核发送间接事务. 当 L1 cache 发生驱逐时采用写回策略(WB),如图 1(c)所示,把脏的数据 Data 写入到 LLC 中,并返回 Ack 消息. 图 2(a)(b)分别展示了 L1 cache 控制器和 LLC 控制器的状态转换图. 图 2(a)左半部分 I→IV→V 展示了 L1 cache 发生读写缺失,发送了 GetS/GetX 消息,然后接收到数据 Data 的状态转换,V→V 展示了读写命中的状态转换. 图 2(a)右半部分 V→VI→I 展示

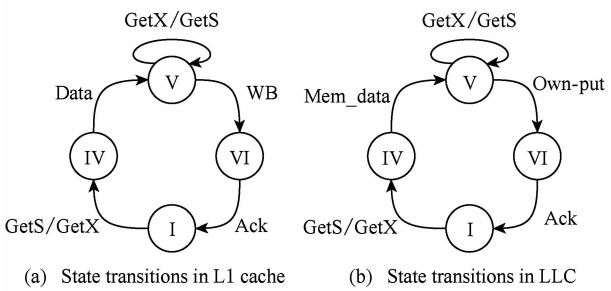

Fig. 2 VISU: State transitions of private data

图 2 VISU 私有数据协议状态图

了 L1 cache 的驱逐,L1 发送 WB 事务把脏的数据写入到 LLC 中,然后接收到 Ack 消息的状态转换. 图 2(b)左半部分展示了 LLC 控制器响应 L1 cache 的 GetS/GetX 消息,然后进行 I→IV→V(LLC 无数据时,从内存中读取并返回 Mem_data 数据)或者 V→V 的状态转换. 图 2(b)右半部分则是 LLC 发生驱逐时,把脏的数据(Own-put)写入到内存中并返回 Ack 消息的状态转换. 整个协议只包含 2 种稳定状态 V 和 I、2 种中间状态 VI 和 IV,并且不需要目录和间接事务.

② VISU 协议针对共享数据的处理. 为了满足一致性的定义,一致性协议必须对写操作立即响应,失效其他核 cache 备份,返回最新的数据. 因为没有了目录记录数据的共享者和拥有者,VISU 协议对于共享数据的处理就要确定最近写入数据的位置并失效过期的备份,所以 VISU 协议添加了一个额外的自更新操作,用以自动更新过期的备份. 相比于私有数据协议,共享数据协议用写直达操作代替写回操作,保证在 LLC 中始终存放最新的数据,从而使得共享数据协议与私有数据协议一样的简单,没有目录和间接事务.

图 3 展示了共享数据的读(Rd)、写(Wrt)、写直达(WT-timeout)、同步点(synchronization, Sync)四种事务.

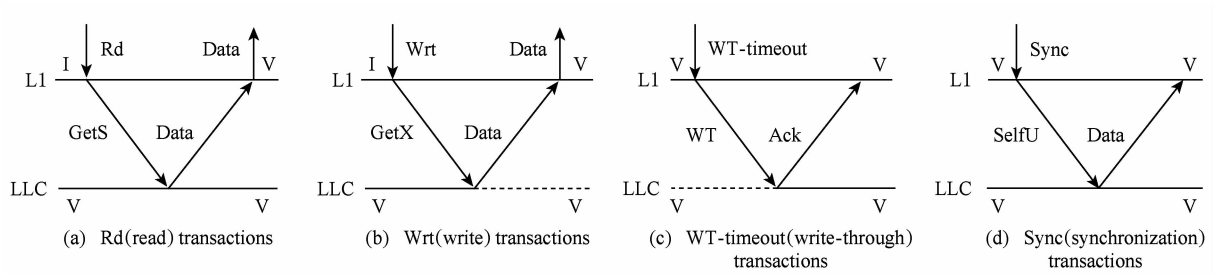

Fig. 3 VISU: Rd, Wrt, WT-timeout and Sync transactions of shared data

图 3 VISU 共享数据读(Rd)、写(Wrt)、写直达(WT-timeout)、同步点(Sync)事务

如图 3(a),首先对于读缺失,L1 cache 发送 GetS 到 LLC 中获取数据 Data. 对于写缺失,如图 3(b),L1 cache 发送 GetX 到 LLC 获取数据,这部分与私有数据协议相同. 不同的是在写操作之后,共享数据协议要进行写直达(WT)操作,把写入的数据及时地写直达(WT)到 LLC 中,这样保证 LLC 一直拥有最新的数据. 为了降低写直达产生的大量报文,如图 3(c)所示,协议采用了延时写直达(WT-timeout)的方式. 最后为了更新过期的备份,协议利用 DRF 编程模型识别同步点,在同步点(Sync)进行自更新操作(SelfU). 如图 3(d)所示,L1 cache 发送 SelfU 到 LLC 中,LLC 返回最新的数据,主动地自更新操作消除了间接的失效.

1.3 VISU 协议正确性论证

本文 1.1 节引入了描述 cache 一致性协议正确性的 2 个条件:单写多读 SWMR 和数据最新(data value invariant),这里我们将论述在 DRF 编程模型下 VISU 协议满足这 2 个条件,即论证协议的正确性.

首先私有数据被单一处理器访问,这些数据不存在 cache 一致性问题. 故 VISU 对于私有数据采用了类似于单核处理器的 cache 写回(write-back)的访问方式. 这满足单写多读和数据最新(data value invariant)条件,即满足 cache 一致性.

其次对于共享数据的访问,对比目录协议,VISU 协议需要论证没有目录和间接失效事务的情况下,如何及时地失效过期的备份和保证读到最新的数据.

1) DRF 编程模型中不同线程对同一个地址的读写之间必然存在同步操作,即在每一个写时期(写操作开始直到遇到同步点)不存在其他线程的读写操作. 例如图 3(b)(c)中 LLC 的虚线段就不存在其他线程针对同一地址的读写操作. 利用写直达方式,每次写操作都能及时把最新的数据写入到 LLC 中,满足了定义中的单写多读的条件.

2) 因为不同线程对同一地址的读写操作(包括读写、写写、写读)之间必然存在同步点,每个同步点都要进行自更新操作. 这样就能及时地更新过期的备份,保证在每个时期开始阶段存储单元的值和最近一次读写时期结束阶段的值(最新写入的值)相同. 因此满足 cache 一致性定义的数据最新原则.

综上所述,对比 cache 一致性定义的 2 个条件,VISU 协议在 DRF 编程模型下满足一致性.

2 VISU 协议的具体实现方案

VISU 协议的实现主要有 3 个部分:数据的划分和切换机制、协议中同步原语的支持 、同步点共享数据的自更新.

具体的设计方案如图 4 所示,从右半部分可以看出,当访存请求发生 TLB 缺失时,就进行页面粒度的数据私有和共享的划分(下文将具体描述). 图 4 左半部分是根据私有共享的划分执行协议,然后进行写直达写回策略的动态切换. L1 cache 发生缺失,就根据数据的私有和共享分别采用 VISU 协议中私有和共享数据处理的 2 个部分发送相应的请求,最终完成数据访存. 下文将具体描述私有共享数据的划分、私有到共享的切换机制、协议中同步原语的支持、同步点共享数据的自更新 4 个部分的实现.

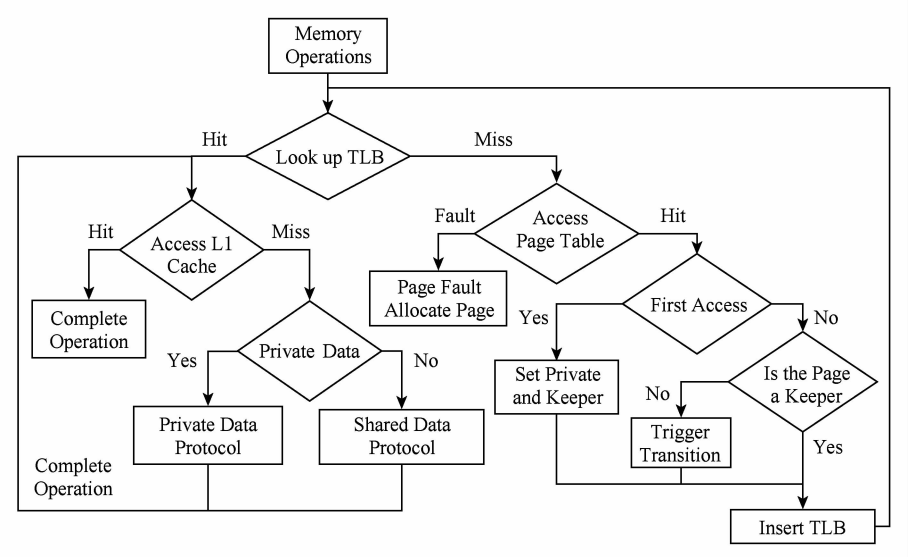

Fig. 4 The block diagram of the action in VISU

图 4 VISU 协议实现框图

2.1 私有共享数据的划分

数据的私有和共享划分,成为了研究者研究的热点. 部分研究工作利用硬件机制进行私有和共享的划分,这些方案虽然能够进行细粒度的数据划分,但是造成了大量的存储开销. 有的研究者利用编译器进行划分,但由于在编译阶段较难判断数据的私有和共享,这种方案实现较为复杂. 本文采用操作系统(OS)辅助进行私有共享数据的划分,Cuesta 等人[2]也采用了类似的划分方法. 这种方案利用 TLB 表项和页表项(page table entry)存储信息. 这是一种页面粒度的私有共享划分方案,当一个页面中的某一个数据共享时,整个页面都标记为共享.

首先对于每一个访存请求,处理器会先查找 TLB 表项,进行虚实地址转换. 每一个 TLB 表项主要由虚拟地址和物理地址 2 部分组成,再加上一些页面属性的标记位. TLB 表项中往往预留一些标志位没有被使用从而允许在 TLB 表项中添加一个私有共享的标记位 P,因此不需要额外的硬件开销. 同理在页表项中添加 P 域和 Keeper 域,分别标记页面是否共享和页面的共享者(第 1 次访问这个页面的 CPU),如图 5 所示.

当 TLB 表项缺失时,TLB 表项和页表项中的域就需要进行处理. 处理方法如图 4 中右半部分所示,访存请求因 TLB 表项缺失进行页表的访问,如果发生缺页,就造成系统异常,操作系统会进行相应的缺页错误处理并分配新的页面. 如果页面被首次访问并且没有缓存在其他 TLB 中,操作系统设置新分配的页面为私有(P 域设为 1)页面且记录第 1 个

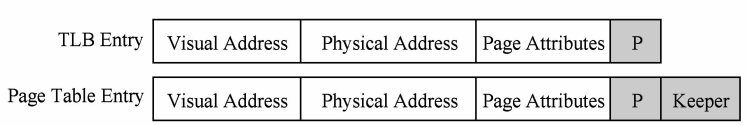

Fig. 5 Format of TLB entry and page table entry

图 5 TLB entry 和页表项的组成

访问的核为 keeper;否则就判断页面的私有和共享的状态,若私有页面且其被不同的核访问,触发私有到共享的切换(下文将详细说明切换机制),切换完成后将页面置为共享(P 域设为 0),最后将设置好标志位的表项添加到 TLB 中.

2.2 私有到共享的切换机制

TLB 缺失就进行页面的访问,当某个核首次访问一个页面时,页面被置为私有页面且访问的核作为页面的拥有者(Keeper). 整个页面里的数据为私有数据,这些数据在 L1 cache 中采用写回的方式. 第 2 个核访问这个页面,发现页面被 Keeper 设置为私有(P 域为 1),这时触发切换机制,向 Keeper 核发送中断,完成中断后设置页表项中的页面为共享(P 域设为 0). Keeper 核接收到中断就更新这个页面的 TLB 表项(设置为共享,P 域设为 0),把 L1 cache 中脏的数据写入到 LLC 中. 完成切换后,所有核访问这个页面都会发现页面共享且 Keeper 核已经将最新的数据写入到 LLC 中,这样页面的数据就切换成了共享数据.

2.3 协议中同步原语的支持

同步机制通常是以用户级软件例程的方式实现,这些例程依赖于硬件提供的同步语句. 在多处理器中实施同步所需要的是一组能够以原子方式读取修改和写入存储单元的硬件原语,例如"Test&Set"或者"Compare&Swap". 如果测试(test)或者比较(compare)一个条件满足,那么核就竞争到一个存储单元,执行原子的读修改写操作;否则在 L1 中对这个存储单元的备份进行旋转,直到存储单元的数据被其他核修改,即条件被改变.

在 VISU 协议中没有间接的失效操作. 一个核结束了敏感区,释放了一个条件,在协议上不会去通知正在旋转的核条件的改变,从而难以完成同步操作. 为此 VISU 一致性协议针对读-修改-写(RMW)的原子事务,采用重新读取 LLC 数据的方式(因为存在多个核访问一个数据,所以数据必然共享,LLC 有最新的数据),让旋转的核能够感知条件的改变. 具体协议如图 6 所示,不妨设条件满足时数据为 0,条件不满足时数据为 1,$L1_0$ 遇到 RMW 原子指令,不管当前是有效还是无效状态(I/V),都发送 GetX 消息到 LLC,LLC 接收到 GetX 就阻塞 LLC,防止原子指令被打断. $L1_0$ 获得数据 Data(0)即条件满足(锁可用)进入修改写入状态 $R(0)W_0(1)$,发送 WT_lock(1),写直达数据 1 占用这个锁,并释放 LLC 的阻塞. 此时程序进入敏感区(critical section)工作,直到释放条件 $W_0(0)$. L1 发送 WT_unlock(0)操作写入 0 到 LLC 中,使条件重新可用. 其他核执行 RMW 时,同样发送 GetX 消息,例如图 6 中 $L1_1$,发现 LLC 被阻塞,则进入 LLC 的等待队列. 当 LLC 不阻塞时,$L1_1$ 获得数据 Data(1),发现条件不满足,$L1_1$ 不断旋转,直到竞争到条件满足读到 Data(0),进入敏感区. 这种绕过 L1 中的数据,直接访问 LLC 的方式,造成了一定的性能损失,但同步操作在整个程序中出现的频率很小,整体性能损失依然可以被接受,第 3 节的实验评估也验证了这一点. VISU 协议通过这种方式实现了对同步原语的支持.

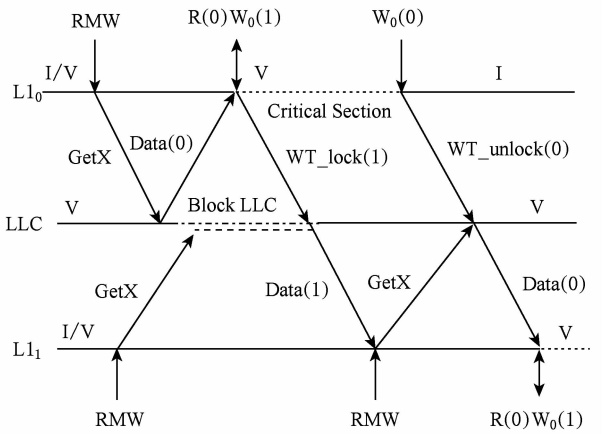

Fig. 6 The atomic RMW transaction of shared data in VISU

图 6 VISU 中共享数据的原子操作(RMW)协议

2.4 同步点共享数据的自更新

在一致性协议中,监听协议通过广播消息作废所有过期的备份,而目录协议通过目录节点间接转发消息失效过期的备份. 之前有研究者采用了自失效操作,即每个核在同步点失效各自 L1 cache 中过期的备份来保证协议的一致性,降低协议的复杂性. 共享数据被多个核使用,很大程度上可能被多次地读写. 如果每一个同步点都采用自失效操作来失效所有的共享数据,无疑会大大增加共享数据的 cache 失效率. 基于此,本文提出了在同步点进行自更新操作. 自更新操作不仅能够有效地降低共享数据 cache 的失效率和提高性能,而且能够起到和自失效操作相同的作用,保证协议的正确性. 每一个进入 L1 cache 的数据都会根据 TLB 表项中的私有共享标志位 P 在 cache line 中设置共享和私有. 基于这些私有共享标记位,当程序识别到同步点时,会完成共享数据从 LLC 到 L1 cache 的更新. 考虑到随着程序的运行,共享数据的数量会不断地增加,为此本设计

中设置了一个阈值,当共享数据数量大于这个阈值时,L1 cache 中就采用自失效操作失效共享数据以防止共享数据的累增.

2.5 实现中优化设计

1) 写直达的粒度优化. VISU 是基于数据无竞争的情况下满足一致性,但是软件的 DRF 并不以 cache line 为粒度. 考虑 2 个核在 L1 中同时拥有一个 cache line 的数据,当这 2 个核同时对这个 cache line 中不同的字(word)进行写操作时,在软件层上这 2 个写操作并不违反 DRF 原则,因此这 2 个并发的写操作之间不存在同步语句. 如果对这种情况不加以处理,最终基于 cache line 粒度的协议延时写直达就会交错覆盖相互的值. 考虑到这个问题,设计中采用了基于 word 粒度的写直达方式,在 L1 cache line 中针对每一个 word 设置 1 个 dirty 位. 在写直达到 LLC 时,不同的核只写入其修改的字(word)而不影响其他字.

2) 直接内存处理(direct memory access, DMA)协议设计. DMA 是现代处理器的重要特色,在协议上也必须对其有相应的支持. DMA 直接使用物理地址进行数据的搬移,不需要经过 TLB 的虚实地址转换. 因此 VISU 协议中的私有和共享的划分对其并不适用,故此设计采用了广播 DMA 请求的方式,失效 L1 中过期的数据,写回脏的数据到 LLC 中. 上述操作完成后,DMA 访问 LLC,获取和写入数据,完成读取和写入操作.

3 实验结果与分析

3.1 实验环境

为了评估 VISU 协议的性能,实验采用 GEM5[1] 全系统模拟器模拟,搭载 Linux 2.6.22 操作系统,采用 GARNET[14] 仿真片上互联网络. 同时利用 CACTI 6.5[15] 工具采用 32 nm 的技术工艺对 cache 的开销进行评估. 设计中仿真 8 核的片上多核处理器(chip multiprocessors, CMP)结构,运行 SPLASH-2 中的测试程序. SPLASH-2 程序分成了 kernels 和 apps 两类程序,实验中分别选取了这 2 类中的 FFT(64 KB complex doubles),LU(LU-contiguous_block, 512×512 matrix),LU-Non(LU-non_contiguous_block, 512 × 512 matrix)和 Water-Nsq(Water-nsquared,512 molecules)并行测试程序进行测试. 模拟器中主要的参数如表 1 所示:

Table 1 Configuration of System

表 1 系统参数配置

Modules	Configuration
Processor Frequency/GHz	1
Cache Line Size	64 B
Page Size	4 KB
Delay Timeout/cycle	500
L1 I&D Cache	32 KB,8-way
Shared L2 Cache	4 MB 512 KB/tile 8-way
Network Topology	2-dimensional Mesh(4×2)
Routing	Deterministic *X-Y*
Garnet-network	Fixed

实验过程中通过识别操作系统和并行程序中的同步点(旋转锁、栅栏、中断)进行自更新操作. 利用 GEM5 全系统模拟器,仿真整个并行程序,统计程序并行阶段的数据. 实验中仿真了 3 种一致性协议:1)MESI 目录协议,该协议由 2 级 cache 组成,实时存储着目录信息. MESI 目录协议根据目录记录的信息发送消息失效所有的共享者(写缺失)或者发送消息获取最新的数据并降级存在的 E/M 状态(读缺失). 2) VIPS 协议,这种协议由 Ros 等人[12] 提出. VIPS 协议在 L1 中也只有 2 种稳定的状态,与本设计中的 VISU 设计类似,但 VIPS 协议在同步点上采用的是自失效操作. 采用同步点自失效的方法保证了协议的一致性,但一定程度上增加了 L1 cache 的缺失率. 3)本文提出的 VISU 协议. 该协议在同步点采用自更新的方式,即在同步点对共享数据进行自更新,提高 cache 的命中率,从而提高系统性能.

在实验过程中,延时写直达(同步点之前)的延时时间设为 500 cycle. 写直达延时主要是因为每次写一个数据,就很可能会访存相应 cache line 的其他字(word)的数据. 延时写直达可以合并写操作,然后一次性写直达到 LLC,降低由写直达造成的大量报文. 但是延时写直达必须在下一个同步点之前完成,所以这个延时时间影响着协议的正确性和系统性能.

3.2 程序请求的分类

VISU 协议动态识别程序中的私有和共享数据,采用 2 种不同的方法进行处理. 实验中运行了 FFT,LU,LU-Non,Water-Nsq 这 4 个程序并记录了并行阶段这 4 个程序对私有数据和共享数据发送请求的数量,把请求分成了 4 类:取指请求、对共享数据的请求、对私有数据的请求以及同步相关的请求,如图 7 所示:

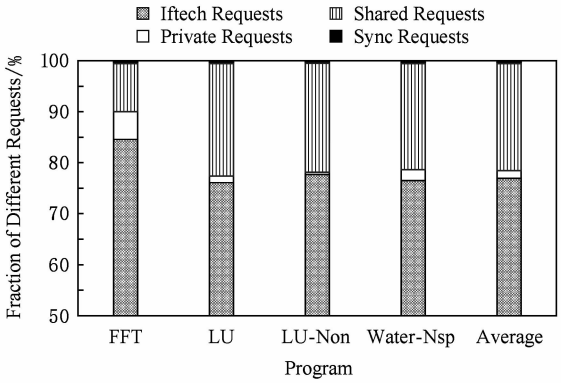

Fig. 7 Fraction of different requests in parallel programs

图 7 并行程序中各类请求的比例

一般情况下,取指请求是一类只读请求,在 VISU 协议中采用写回的方式进行处理,不进行私有和共享的划分. 从图 7 可以看出,私有数据请求比例在 FFT 中最大,占程序总请求的 5.1%. 若把私有数据请求和取指请求均作为私有请求处理(二者采用相同的私有数据协议),则私有请求数量占有比例高达 90.1%. 而在 LU 和 Water-Nsq 程序中私有请求占比较少(分别占有 1.2%和 1.8%)、共享请求占比大,这影响着这 2 个程序在 VISU 协议中的性能. 整体上来看,私有请求(包括取指请求)平均占整个并行阶段请求的 77.2%. 在 4 个程序中(FFT, LU, LU-Non, Water-Nsp)同步相关的请求所占比例依次为 0.11%, 0.10%, 0.08%, 0.22%, 其中 Water-Nsp 相对来说比例较高,4 个程序同步请求平均占比 0.14%.

3.3 VISU 协议运行性能

为了分析 VISU 协议性能,实验测试了 MESI, VIPS, VISU 这 3 种协议,并记录了程序并行阶段的运行时间,如图 8 所示.

实验结果以 MESI 协议的运行时间进行归一化. 平均来看,无目录、有 2 个稳定状态的 VISU 协议获得了与有目录、3 个稳定状态的 MESI 协议几乎一样的性能. 从 FFT 程序中可以看出,VISU 性能较之 MESI 协议有 11%的性能提升,一方面来源于私有数据占有比例高;另一方面来源于同步操作,在 VISU(VIPS)协议中对于同步操作进行自更新(自失效),这是一类为了保证正确性但开销大的操作,但在 FFT 中所占比例很少. Water-Nsp 的同步操作占有的比例较大,同时私有请求比例较少,所以 VISU 协议的整体性能相比于 MESI 协议有了 8%的性能损失. 在同步点比例大时,VISU 协议相对

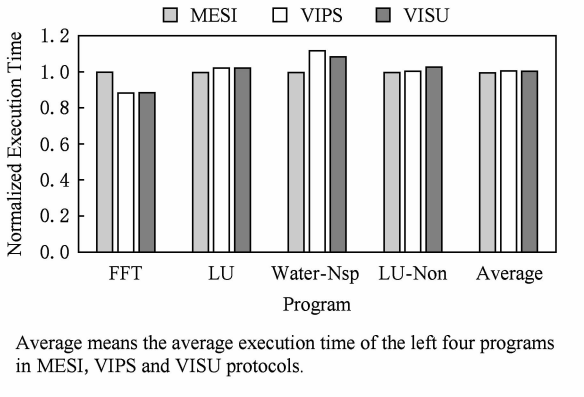

Fig. 8 Normalized execution time

图 8 归一化的运行时间

VIPS 协议能够降低由于同步点导致的大量 cache 失效,从 Water-Nsp 程序上可以看出,相对 VIPS 协议,VISU 有了 4%的性能提升. 整体上来看,简单地更新有限共享数据的 VISU 协议性能略高于 VIPS 协议(平均 0.2%的性能提高).

3.4 VISU 的开销

在存储开销方面,表 2 为 3 种协议的特征对比,MESI 目录协议采用全映射(full-map)目录. 图 9 展示的是 N 节点系统的 MESI 协议基本的目录表项,其中 2 个主要的存储开销:每个数据 block 的拥有者(owner)和数据 block 的共享列表(sharer list),每个表项都需要增加 $\mathrm{lb}\,N$ 位和 N 位. 这些存储开销严重地影响目录协议的可扩展性. 在 MESI 目录协议中,目录信息存储在 LLC 的 tag 中,实验中测试了 3 种协议 LLC tag 的面积开销,如表 2 所示. VIPS 和 VISU 协议没有目录,从而降低了 25.4%的面积开销.

Table 2 Comparison of Three Protocols

表 2 3 种协议对比

Protocol	Directory	LLC Tag Area/mm^2	Invalidation	L1 States	
				Stable	Transient
MESI	Full-map	0.127	Multicast	4	11
VIPS	No	0.0952	Self-invalidation	2	2
VISU	No	0.0952	Self-updating	2	2

cache 一致性协议的验证问题是 CMP 设计中的重要课题,在验证开销方面,VISU 与 VIPS 协议分别采用自更新和自失效的方式作废过期的备份,这是 2 种 1 对 1 的方式,而 MESI 目录协议采用的是利用目录的共享列表进行多播,如表 2 所示.

此外目录协议通过目录向数据拥有者(owner)发送数据请求,增加了额外的间接事务,而 VISU 和

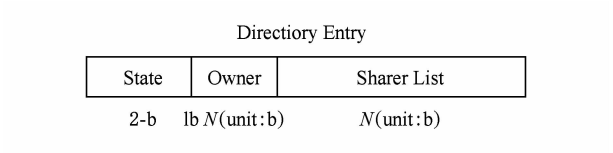

Fig. 9 Directory entry for a bock in a system with N nodes

图 9 N 个节点系统中 block 的目录表项

VIPS 协议没有间接访问. VISU 协议利用私有和共享数据的划分,把协议分成了独立的 2 个部分,可分别进行验证. 同时 VISU 协议只有 2 个稳定状态:有效(valid)和无效(invalid),在 L1 cache 中只有 2 个中间状态(IV,VI),相对于 MESI 协议有 4 种稳定状态、在 L1 cache 中有 11 种中间状态和各种间接事务,VISU 和 VIPS 协议在验证方面具有显著的优势.

3.5 VISU 中延时写直达的影响

为了进一步说明写直达中延时时间对于 VISU 协议的影响,实验中分别测试了 100 cycle,500 cycle,1 000 cycle 延时对于测试程序报文数量的影响. 如图 10 所示:

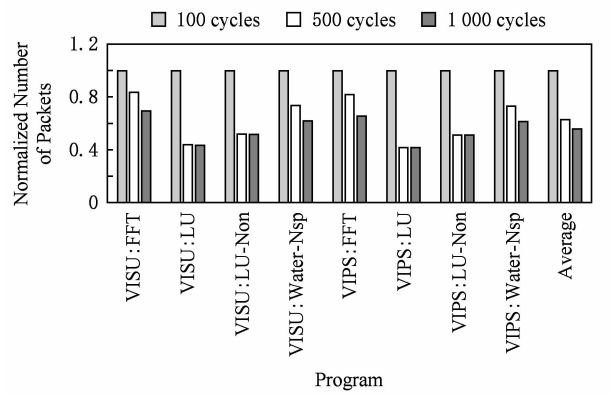

Fig. 10 Number of packects in the VISU and VIPS

图 10 VISU 和 VIPS 写直达延时时间与报文数量

采用延时写直达,能够利用局部性原理,合并多次对同一个 cache line 的写操作,减少写直达的次数,降低整个系统的报文数量. 但延时写直达操作,必须保证 VISU 协议的正确性和系统性能,确保写入的数据能够及时对其他核可见. 从图 10 可以看出,当延时从 100 cycles 增加到 500 cycles 时,报文数量显著减少,平均减少了 30.7%. 但从 500 cycles 增加到 1 000 cycles,报文数量整体减少了 6.8%,其中 LU 和 LU-Non 这 2 个程序报文数量几乎没有减少. 因此在 VISU 协议中设置延时写直达的时间为 500 cycle.

3.6 VISU 协议对 cache 缺失率的影响

VISU 协议与 VIPS 协议的主要区别在于采用自更新的方式提高 L1 cache 的命中率. 对于同步点采用自更新还是自失效,主要取决于 2 个同步点之间的数据是否会被再一次访问. 图 11 描述的是 VIPS 和 VISU 协议的 L1 cache 失效率,以 VIPS 协议的失效率进行归一化.

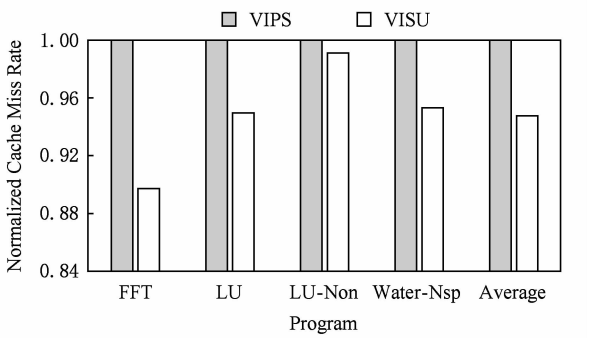

Fig. 11 Normalized cache miss rate in the VISU and VIPS

图 11 自更新 VISU 协议与自失效 VIPS 协议缺失率

由于协议中自更新和自失效的数据都是共享数据,共享数据被多个核访存,很可能会被再一次访问到. 同时为了解决在自更新的过程中共享数据不断增加的问题,在 VISU 协议中采用预设一个阈值,当共享数据的数量超过这个阈值时,超过的部分采用自失效,防止共享数据的累增. 当共享数据的数量少于这个阈值时,则采用自更新方式. VISU 协议采用最后访问的共享数据最先被自更新的策略,设置一个阈值 x,对最后被访问的 x 个共享数据进行自更新,对其他的共享数据采用自失效,在本实验中设置 $x=50$. 利用在同步点进行自更新的方式可以明显地降低 cache 失效率,自更新的 VISU 协议相比 VIPS 协议平均降低了 5.2%的 L1 cache 失效率.

4 结束语

本文针对于共享存储中的 cache 一致性协议进行了简化,提出一种没有目录和间接访问、没有众多一致性状态和竞争的 cache 一致性协议 VISU. VISU 协议的关键:1)区分私有和共享数据,对私有数据采用写回策略,对共享数据采用写直达策略;2)利用 DRF 编程模型中的同步点进行自更新. 在 VISU 协议中,自更新是一种高效处理过期备份的方式. 与一直以来获得广泛使用的目录式和监听式协议相比,VISU 协议大大简化了复杂性,解决了

一直以来困扰目录协议可扩展性的目录开销问题. VISU 协议不仅不需要像监听协议一样广播大量的请求,并且还适用于 mesh 和 torus 等无序网络.

通过仿真与分析,相对于复杂的 MESI 协议,简单的 VISU 协议具有与之相当甚至更优的性能. 接下来的工作将会针对 VISU 自更新操作,优化共享数据,提高自更新操作的效率. 并进一步将 VISU 协议扩展到层次集群式的 cache 结构(hierarchical clustered cache)上,提高 VISU 协议的可扩展性.

参 考 文 献

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

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

[3] Kaxiras S, Keramidas G. SARC coherence: Scaling directory cache coherence in performance and power [J]. IEEE Micro, 2010, 30(5): 54-65

[4] Menezo L G, Puente V, Gregorio J A. Flask coherence: A morphable hybrid coherence protocol to balance energy, performance and scalability [C] //Proc of the 21st IEEE Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2015: 198-209

[5] Hossain H, Dwarkadas S, Huang M. POPS: Coherence protocol optimization for both private and shared data [C] //Proc of Int Conf on Parallel Architectures and Compilation Techniques. Los Alamitos, CA: IEEE Computer Society, 2011: 45-55

[6] Hou Fangyong, Gu Dawu, Xiao Nong, et al. Performance and consistency improvements of Hash tree based disk storage protection [C] //Proc of IEEE Int Conf on Networking, Architecture, and Storage. Los Alamitos, CA: IEEE Computer Society, 2009: 51-56

[7] Qian Xuehai, Sahelices B, Qian Depei. Pacifier: Record and replay for relaxed-consistency multiprocessors with distributed directory protocol [J]. ACM SIGARCH Computer Architecture News, 2014, 42(3): 433-444

[8] Huang He, Liu Lei, Song Fenglong, et al. Architecture supported synchronization-based cache coherence protocol for many-core processors [J]. Chinese Journal of Computers, 2009, 32(8): 1618-1630 (in Chinese)
(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)

[9] Li Gongming. Cache coherence techniques for chip multiprocessor architecture [D]. Hefei: University of Science and Technology of China, 2013 (in Chinese)
(李功明. 片上多核处理器体系结构中 Cache 一致性模型研究[D]. 合肥: 中国科学技术大学, 2013)

[10] 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)
(张俊, 田泽, 梅魁志, 等. 基于节点预测的直接 cache 一致性协议[J]. 计算机学报, 2014, 37(3): 700-720)

[11] Kim D, Ahn J, Kim J, et al. Subspace snooping: Filtering snoops with operating system support [C] //Proc of the 19th Int Conf on Parallel Architectures and Compilation Techniques. New York: ACM, 2010: 111-122

[12] Ros A, Kaxiras S. Complexity-effective multicore coherence [C] //Proc of the 21st Int Conf on Parallel Architectures and Compilation Techniques. New York: ACM, 2012: 241-252

[13] Sorin D, Hill M, Wood D. A Primer on Memory Consistency and Cache Coherence [M]. Williston, VT: Morgan & Claypool Publishers, 2011

[14] Agarwal N, Krishna T, Peh L S, et al. GARNET: A detailed on-chip network model inside a full-system simulator [C] //Proc of Int Symp on Performance Analysis of Systems and Software. Piscataway, NJ: IEEE, 2009: 33-42

[15] Muralimanohar N, Balasubramonian R, Jouppi N P. Architecting efficient interconnects for large caches with CACTI 6.0 [J]. IEEE Micro, 2008, 28(1): 69-79

He Ximing, born in 1991. Received his MSc degree in computer science and technology from the National University of Defense Technology (NUDT) in 2018. His main research interests include network on-chip and cache coherence.

Ma Sheng, born in 1986. Received his BSc and PhD degrees in computer science and technology from the National University of Defense Technology (NUDT) in 2007 and 2012, respectively. He visited the University of Toronto from Sept. 2010 to Sept. 2012. His main research interests include on-chip networks, SIMD architectures and arithmetic unit designs.

Huang Libo, born in 1983. Received his BSc and PhD degrees in computer engineering from the National University of Defense Technology (NUDT) in 2005 and 2010, respectively. His main research interests include VLSI design, computer architecture on-chip communication, hardware/software co-design.

Chen Wei, born in 1982. Received her BSc, MSc and PhD degrees in computer science and technology from the National University of Defense Technology (NUDT) in 2004, 2006 and 2010, respectively. She visited the University of Singapore Nanyang Technological University form Nov. 2006 to Nov. 2007. Her main research interests include high performance computer architecture and microarchitecture.

Wang Zhiying, born in 1956. Received his PhD degree in electrical engineering from the National University of Defense Technology (NUDT), China, in 1988. His main research interests include computer architecture, computer security, VLSI design, reliable architecture, multicore memory system, and asynchronous circuit.