硬件结构支持的基于同步的高速缓存一致性协议

黄 河[1),2)] 刘 磊[1),2)] 宋风龙[1),2)] 马啸宇[1),2)]

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

[2)](中国科学院研究生院 北京 100049)

摘 要 共享存储系统中如何高效地实现高速缓存一致性是体系结构设计面临的一个关键问题和难点问题.已有的基于目录的协议存在难于实现、验证复杂和存储空间开销大等问题.面向片上众核处理器,文中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,而是通过使用 bloom-filter 表示一致性信息,并在并行程序中的同步点维护高速缓存一致性.与现有的基于目录的高速缓存一致性协议相比,该方案可以降低目录协议的实现、验证复杂度.用 SPLASH-2 测试程序集评估表明,基于同步的协议可以获得与基于目录的协议相当的性能.

关键词 高速缓存一致性;存储一致性模型;多核处理器;共享存储系统

中图法分类号 TP303 **DOI 号**: 10.3724/SP.J.1016.2009.01618

Architecture Supported Synchronization-Based Cache Coherence Protocol for Many-Core Processors

HUANG He[1),2)] LIU Lei[1),2)] SONG Feng-Long[1),2)] MA Xiao-Yu[1),2)]

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

[2)](*Graduate University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract The efficient support of cache coherence is extremely important to design and implement many-core processors. This paper proposes a synchronization-based coherence protocol to efficiently support cache coherence for shared memory of many-core processors. The unique feature of the scheme is that it doesn't use directory at all. Inspired by scope consistency memory model, the protocol maintains coherence at synchronization point. Within critical section, process cores record write sets (which lines have been written in critical sections) with bloom-filter functions. When the core releases the lock, the write set is transferred to a synchronization manager. When another core acquires the same lock, it gets the write set from the synchronization manager and invalidates stale data in its local cache. The scheme is evaluated using programs from SPLASH-2 benchmark. The results show that synchronization-based protocol can achieve similar performance in cost-effective way compared to a directory-based protocol that requires large amount of hardware resources and huge design verification effort.

Keywords cache coherence; memory consistency; many-core processors; shared memory system

---

收稿日期:2008-06-10;最终修改稿收到日期:2009-05-25.本课题得到国家自然科学基金重点项目(60736012)、国家"九七三"重点基础研究发展规划项目基金(2005CB321600)资助.**黄 河**,男,1977 年生,博士研究生,主要研究方向为处理器微结构、片上多核处理器体系结构、操作系统及 ASIC 后端设计.E-mail: huangh@ict.ac.cn.**刘 磊**,男,1981 年生,博士研究生,研究方向为处理器微结构和低功耗集成电路设计.**宋风龙**,男,1980 年生,博士研究生,研究方向为片上存储系统.**马啸宇**,男,1984 年生,博士研究生,研究方向为片上多核处理器体系结构.

1 引　言

设计片上多核处理器(Chip MultiProcessor,CMP)时,如何高效地维护片内高速缓存一致性是结构设计者面对的一个关键问题和难点问题.现有片上多核处理器和 CC-NUMA(Cache Coherence Non Uniform Memory Access)并行系统中使用的基于侦听的协议和基于目录的协议由于扩展性和效率问题并不适用于未来的众核处理器(单芯片内集成数十个乃至更多的处理器核的片上多核处理器).侦听协议由于需要使用总线互连提供的广播功能,使得其只适用于规模较小的 CMP 或 SMP 系统,而不能用于众核处理器系统.在未来的众核处理器中,连接各处理器核的片上网络很可能采用点对点链路构成类似网孔(mesh)的拓扑结构.这种拓扑结构与 CC-NUMA 并行系统的互连拓扑结构有类似之处.采用基于目录的协议实现的 CC-NUMA 系统中,已经有规模达到数百个处理器节点的系统[1],基于目录的协议的扩展性得到确认.但是,如果将基于目录的协议应用到实现在单芯片内的众核处理器,目录协议由于设计实现复杂、存储空间开销较大、以高速缓存行(cache line)粒度维护一致性导致的一致性消息较多等原因,并不一定适用于众核处理器.具体来说,由于目录协议以高速缓存行为单位维护一致性,通常每一个高速缓存行都要用一个宽度为系统中处理器数目的位向量(bit vector)来标识哪些处理器中有此高速缓存行的备份.假设众核处理器中有 p 个处理器核,而每个处理器核管理 c 个高速缓存行,这样一个处理器核中维护共享信息的位向量有 $c\times p$ 位,p 个处理器核共有 $c\times p^2$ 位.可见,位向量的存储开销与众核处理器中处理器核数的平方成正比,不利于处理器核数的扩展.在众核处理器中,常见的运行方式是同时运行多个多线程程序.如图 1 所示,对于应用程序本身的全局数据只需要维护同一程序内各线程所在的处理器核之间的高速缓存一致性;在不同程序间,不需要维护属于各程序的全局数据的一致性,只需要维护共享的操作系统内核或运行时系统中数据的一致性.而目录协议中采用的位向量覆盖了片内所有处理器核,带来了存储空间上的浪费.有限指针、粗粒度目录等优化存储空间开销的方案加剧了目录协议的复杂度并且对性能带来损失.基于目录的协议由于存在很多中间状态和边界情况以及要处理死锁等复杂问题,导致实现及验证的难度非常大[2],甚至超过处理器核中其它大多数模块的设计与验证复杂度[3].

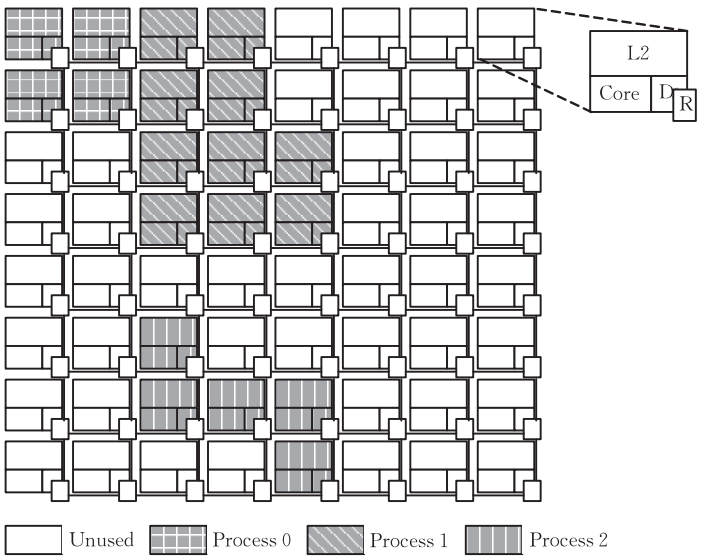


图 1　分片式片上多核处理器

针对以上问题,本文提出一种硬件结构支持的基于同步的高速缓存一致性协议,该方案不使用目录维护一致性信息,在降低设计和验证时的复杂度的同时,可以获得与基于目录的协议相当的性能.

本文第 2 节简介典型的高速缓存一致性协议的相关工作,指出目录协议在众核环境下的协议的开销、可扩展性等问题;第 3 节介绍硬件支持的基于同步的高速缓存一致性协议;第 4 节讨论方案与实现相关的问题;第 5 节对比本方案与目录协议的实现复杂度和存储空间开销;第 6 节给出性能评估结果;第 7 节对本文工作进行总结.

2 相关工作

共享存储多处理器系统中,同一存储单元内容可以被不同的处理器缓存在核内的高速缓存中,并且同一存储单元的变化可能在不同的时刻被不同的处理器接受.为了保证并行程序的正确执行,必须定义系统的存储一致性模型(memory consistency model)并通过高速缓存一致性协议(cache coherence protocol)实现该存储模型.目前,在 CMP 和 CC-NUMA 系统中[1,4-5]通常采用基于目录的 MESI 协议来维护高速缓存的一致性,并实现某种存储一致性模型.如 Stanford DASH 系统[4]采用基于目录协议实现了释放一致性(release consistency)存储模型.在基于目录的协议中,对于每一高速缓存行(cache line)都要有一个位向量来描述共享信息.位向量中的每一位对应一个处理器或一个多处理器节点.当处理器核数达到数百个核的量级时,目录中维护共享信息的位向量需要的存储空间将接近甚至超过高速缓存行数据所占存储空间,导致目录的存储空间利用率较低.基于目录的一致性协议以高速缓存行为单位来维护一致性,对于每个高速缓存行都要执行一遍一致性协议.如对于写操作,当没有写权限时需要查看目录判断是否有其它核共享该高速缓存行,如果有共享则要逐一将其它核中的副本无效掉.所以当处理器核较多时,基于目录的协议在逻辑复杂度和性能上都存在不利于扩展的因素.在众核处理器中,由于目录和协议控制器都集成在片内,相对于多处理器系统,这些问题更为明显.除了扩展性的原因,目录协议在设计和验证时非常复杂.在实际的工程实现时,面临诸如如何避免死锁、如何实现系统规定的访存一致性模型等一系列不可回避且处理繁琐的细节问题[2,4,6].

本文的一致性方案是受软件共享虚拟存储(shared virtual memory)系统[7](又称为软件 DSM 系统)的启发.共享虚拟存储系统在各自独立而通过互联网络松散耦合的系统(例如,由局域网互联的机群系统)中用软件的方法实现共享存储的编程界面.通常的做法是在并行程序的同步点维护共享数据的一致性(这里含义既包括 cache coherence 也包括 memory consistency).代表性的工作包括域一致性存储模型(Scope Consistency)[8].域一致性存储模型以一致性区域(consistency scope)为单位维护高速缓存一致性,一致性区域可以是隐式的、由共享存储并行程序中同步原语界定,也可以由程序员通过显示的指令界定.例如,获取锁和释放锁的操作(lock acquire 和 lock release)既定义了互斥的临界区,也隐式地定义了一致性区域.域一致性存储模型只维护一致性区域内的访存操作的完成次序,而对于一致性区域外的访存操作不做限制.域一致性存储模型利用了共享存储并行程序中同步操作表达的一致性需求,较好地兼顾了可编程性(相对于单项一致性存储模型)与性能.本方案中,用基于同步的协议支持域一致性存储模型.一些软件 DSM 系统支持域一致性存储模型[7],通常的做法是将在临界区内的改写过的页面标识(write-notice)以及页差(diff)记录下来,当释放锁时将 write-notice 发送到锁管理器,将页差发送到页面的 home 节点并合并到内存中.当另一个线程获取锁时,同时得到附着在这把锁上的 write-notice,利用 write-notice 将本地缓存的页面无效掉.这样当在临界区内读取该页面的数据时,由于旧数据已经被做无效处理,所以可以从 home 节点正确取得新数据.在文献[8]提到了通过在网络接口上由硬件结构支持写更新来加速支持域一致性存储模型的软件 DSM 系统,这里的硬件结构支持与本方案在目标与做法上都完全不同.

对于众核处理器结构的研究刚刚起步,其中对于适合于众核结构的高速缓存一致性协议的研究成果十分有限.MIT 的 RAW 处理器集成了 16 个处理器核,各个处理器核共享主存储器.在 RAW 中不由硬件支持高速缓存一致性,而需要程序员或编译器的干预来保证高速缓存一致性.这使得 RAW 的可编程性受到极大限制.RAW 处理器的商品化设计 TILE64 处理器[5]中采用的是避免 cache coherence 问题的方法从而省略了支持高速缓存一致性的硬件结构.TILE64 中的 L2 是分布式 NUCA 的,每个处理器核的 DCache 只缓存本地 L2 管理的数据,而不缓存其它处理器核节点 L2 管理的数据;当发生 DCache miss 时,如果高速缓存行的 home 节点不是自身,则从 remote 节点取回数据后不在 DCache 中留有备份.在这种设计中,不存在 cache coherence 问题,只存在 memory consistency 问题.虽然省略了支持高速缓存一致性的硬件结构可以使设计复杂度大为降低,但是由于不缓存非本地节点的数据,类似 TILE64 的设计在访问片内其它处理器核节点的数据时的开销较大,从而影响整体性能.

文献[9]中的工作与我们的目标是相同的,即在众核处理器中不使用基于目录的协议而通过其它方

式维护高速缓存一致性.在文献[9]中 DCache 可以缓存 home 节点不是本地节点的数据,并且可以以页面为单位动态设置高速缓存行的 home 节点.页面数据所在节点的信息保存在 OS 管理的页表中,使用一个类似于 TLB 的 MAP 表来缓存最近使用的表项.在并行程序的同步点,文献[9]将 MAP 中标记为共享的表项都做无效处理,以此保证旧数据不会被错误使用.文献[9]与我们的设计同样借鉴了软件 DSM 系统在并行程序的同步点维护共享数据的一致性的思想.但是我们的方案与文献[9]完全不同.在功能上,文献[9]在同步点将共享的页面都无效掉,一方面会导致属性为共享但是并未被其它处理器核写过的页面也被不必要地无效掉,另一方面会导致一个页面中并未被其它处理器核写过的高速缓存行也被不必要地无效掉.这虽然不会导致逻辑错误,但是降低了性能.在我们的方案中,很好地避免了这些问题.文献[10]用硬件结构支持由软件实现的目录协议,也即目录协议由运行在处理器核上的软件完成.通过硬件结构支持一致性权限的检测,来降低实现目录协议的软件对系统性能的影响.在文献[10]中使用写权限缓存表来记录已经具有写权限的地址空间,当写操作在写权限缓存表中命中时,表明该处理器核唯一地持有写操作要访问的存储空间,从而可以进行写操作.如果没有写操作缓存表,则对于程序中的每个写操作都需要进行权限检查,通常的做法是每个写操作都触发例外或在生成程序的可执行代码时插入权限检代码,这种开销对性能的影响是显而易见的.很明显,我们的方案与文献[10]虽然目标相同,但是内容与做法都不相同.

Bulk[11]和 BulkSC[12]中利用 bloom-filter[13]记录、检测地址集合间的冲突.本文利用 bloom-filter 表示一致性信息,并据此在并行程序的同步点维护高速缓存一致性.

3　硬件结构支持的基于同步的高速缓存一致性协议

在设计共享存储系统时,采用何种高速缓存一致性协议实现何种存储一致性模型需要兼顾存储模型的性能、协议实现的复杂度与可扩展性、系统的可编程性等因素.在 3.1 节中,首先给出本文方案的研究对象:分片式片上多核处理器;在 3.2 节中,介绍硬件结构支持的基于同步的高速缓存一致性协议;3.3 节介绍关键数据结构 W-set 的表示方法;3.4 节给出硬件结构同步管理器的设计;3.5 节给出一致性信息 W-notice 的管理方法;3.6 节讨论如何支持嵌套的同步操作;3.7 节给出如何实现域值一致性存储模型.

3.1　分片式片上多核处理器

图 1 给出了分片式(tiled)片上多核处理器的结构.其中,每个节点(分片)由处理器核、二级高速缓存和片上网络路由器等模块构成.各节点间通过路由器构成 2-D mesh 互连结构.分片式片上多核处理器是一种较为通用的结构,在多核体系结构的研究中被广泛采用.因此,本文把这种多核结构作为研究对象.

图 1 中各处理器核内的 DCache 和 ICache 是私有的,L2 是分布式共享的,每个节点的 L2 缓存管理由物理地址划分开的一部分存储空间,各处理器核内的 DCache 既可以缓存本地节点 L2 中的数据,也可以缓存 home 节点是其它处理器核的 L2 中的数据.在 L2 中数据块只存在于 home L2 节点.DCache 的写策略是 writeback.高速缓存行在 DCache 中可以是以下 3 个状态之一:INVALID、CLEAN 和 DIRTY.同一数据块可以在不同节点的 DCache 中同时存在,且可以同时为 DIRTY 状态.这是由于我们的协议不是以高速缓存行为单位维护一致性.

3.2　硬件结构支持的基于同步的高速缓存一致性协议

在共享存储并行程序中,程序员用锁和栅障等原语界定临界区以同步对共享数据的访问.例如,锁用来实现对共享数据的互斥访问,栅障用来同步并行程序的各个执行阶段.除了同步语义本身以外,同步原语还表达了对其作用的共享数据的一致性需求.软件 DSM 系统中利用此思想在应用程序的同步点维护高速缓存一致性[7].本文方案借鉴了软件 DSM 系统的做法,在应用程序的同步点维护高速缓存一致性,而不是基于目录的协议以高速缓存行为单位维护一致性.在软件 DSM 系统中通常是以操作系统管理的页面为单位在同步点维护高速缓存一致性.具体的做法是在临界区内记录线程修改过哪些页面,这些修改过的页面号集合记作 write-notice,将修改过的页面的数据部分与初始页面数据的差异记作块差 diff.在应用程序的同步点,释放锁的线程将 write-notice 传递给锁管理器,将页差 diff 传送到各个被修改过的页面所在的 home 节点,并且将 diff 合并到内存中.当另外的线程获得锁时,该线程同时获得 write-notice,并根据 write-notice 将本节

点缓存的旧数据无效掉，当要访问前一个线程更新过的数据时，由于在本节点的缓存中不命中，所以到该页面所在的 home 节点取得新数据. 直接将软件 DSM 系统硬件结构化是非常困难的，一方面以页面为粒度维护一致性由于假共享等问题不利于系统的性能，另一方面难于用硬件记录 write-notice，diff，twin 等复杂的软件数据结构.

本文提出的硬件结构支持的基于同步的协议与软件 DSM 中的同步协议在原理上类似，而在共享块的粒度和 write-notice 的表示与软件 DSM 不同，且避免了 diff 和 twin 等复杂的数据结构. 硬件结构支持的基于同步的协议如下：

(1) 在并行程序的同步点维护高速缓存一致性；

(2) 共享数据块的粒度是 DCache 高速缓存行；

(3) DCache 的写策略是 write-back；

(4) 在并行程序的临界区内，处理器核记录写过的 DCache 行地址集合，记为 W-set；

(5) 当执行 release 操作前，将临界区内写过的 DCache 行写回到共享的 L2，W-set 保存到同步管理器(同步管理器中的 W-set 称为 W-notice)；

(6) 执行 acquire 操作请求锁的处理器核在得到锁权限的同时从同步管理器取得锁对应的 W-set；

(7) 请求锁的处理器核在执行临界区内的访存操作前，用取得的 W-set 将本地 DCache 中相应的行无效掉；

(8) 实现域一致性存储模型或释放一致性存储模型.

为了本文的方案，需要对处理器核访存流水线进行相应的修改. 图 2 给出了在龙芯-2 号处理器核基础上支持同步协议的超标量处理器核访存流水线结构图.

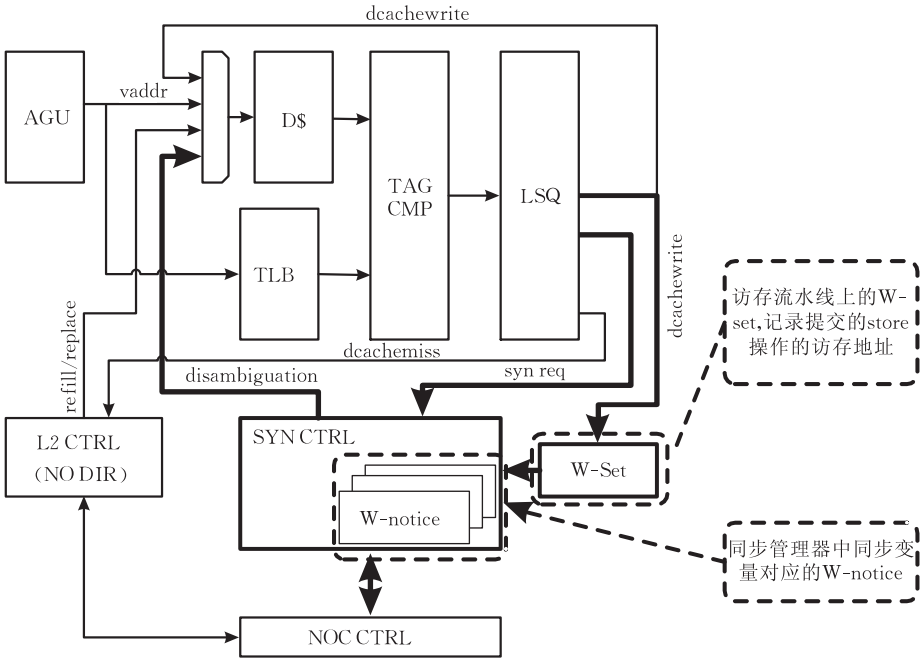


图 2 同步协议的访存流水线

图 2 中粗线框和粗线条是新增的数据通路. 其中访存流水线中的 W-set 用于记录在临界区内提交的 store 指令的高速缓存行地址. 同步管理器中的 W-notice 是各处理器核的 W-set 在执行 release 操作后合并的结果. 同步管理器的功能是对 W-notice 进行管理，在同步管理器设计一节进行详细介绍.

3.3 W-set 的表示

W-set 是高速缓存行地址的集合. 当用硬件结构实现时，W-set 的表示方法应该有以下特点：(1) 能够表示较大的集合；(2) 集合操作的逻辑简单，便于硬件实现. 由于不同的程序在临界区内改写的共享数据集合大小差异较大，如果用固定大小的硬件表来存储修改过的高速缓存行集合，那么需要一个存储空间开销较大的硬件表，并且需要解决硬件表溢出的问题.

本方案中采用 bloom filter[13] 记录 W-set. Bloom filter 是一种用有限资源表示接近于无限集合的工具，同时对集合的并、交、判断一个元素是否

属于集合等操作可以通过简单逻辑运算完成. Bloom filter 是集合的不精确表示,存在本不属于原集合的元素被判定为在 bloom filter 表示的集合中的情况(false sharing,误识),但不存在漏识(属于原集合的元素被判定为不在 bloom filter 表示的集合中). 在本方案中,发生误识只会导致本不必要无效掉的高速缓存行被无效掉,从而导致不必要的 DCache 回填操作. 这会带来一定的性能损失而不会导致逻辑错误.

表 1　W-set 操作

操作	说明
void insert(W-set, addr)	将访存地址 addr 加入到 W-set.
bool membership(W-set, addr)	判断地址 addr 是否属于 W-set.
W-set union(W-set-0, W-set-1)	逻辑'或'操作,将 W-set-0 与 W-set-1 合并.
bool disambiguation(W-set-0, W-set-1)	逻辑'与'操作,判断 W-set-0 与 W-set-1 是否存在交集.

3.4　同步管理器设计

本方案中,并行程序中的锁、栅障等同步操作由 CAS(Compare And Swap)原子指令构成. 与在目录协议中的不同是,本方案中的 CAS 是 uncache 的. 为了支持同步协议,定义界定一致性区间的指令(一致性指令)如表 2、表 3 所示.

表 2　一致性指令

open_scope synid	从同步管理获得同步变量的 synid 的 W-notice,并控制在本处理器核访存流水线开始记录 W-set.
close_scope synid	停止在本处理器核访存流水线中记录 W-set,并将 W-set 发送到同步管理器.

表 3　线程库中的同步函数

acquire:	release:	barrier:
lock(synid);	close_scope synid	close_scope synid
open_scope synid	unlock(synid);	barwait(synid);
		open_scope synid

应用表 2 中的一致性指令可以构成线程库中的 acquire、release 和 barrier 等同步函数,如表 3 所示. 当 close_scope 指令执行时,同步管理器接收 close_scope 指令发送的 W-set,并将 W-set 处理后存储到同步变量对应的 W-notice 中. 对于 open_scope 指令,同步管理器向处理器核返回该同步变量对应的处理器核的 W-notice.

3.5　W-notice 管理

W-notice 是 bloom filter 表示的一致性信息. 维护 W-notice 要满足:

(1)正确性. 执行 open_scope 获得的 W-notice 必须包含此前各个处理器核执行 close_scope 时生成的 W-set,即要保证处理器核在根据 W-notice 无效掉本地 DCache 中相应的高速缓存行后,本地 DCache 中不再包含其它处理器核改写过的高速缓存行;

(2)有效性. W-notice 是由 bloom filter 表示的位向量,要避免由于不及时清除已经使用过的 W-notice 导致 W-notice 累积满从而使得 W-notice 失效,即由于 W-notice 满使得在 open_scope 指令执行时,需要将本地 DCache 完全无效掉的情况.

针对以上两点,同步变量和 W-notice 的组织如图 3 所示.

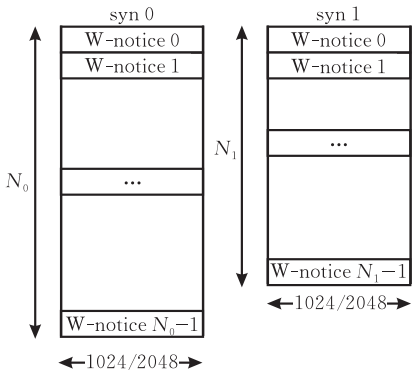


图 3　W-notice 的组织(N_i 为使用同步变量 Syn_i 的处理器核数)

W-notice 处理算法如下:

对于处理器核 i 发出的 close_scope synid 指令,同步管理器将 close_scope 指令携带 W-set 合并到同步变量 synid 除了处理器核 i 本身之外所有其它处理器核的 W-notice 中.

对于处理器核 i 发出的 open_scope synid 指令,同步管理器将同步变量 synid 中处理器 i 的 W-notice 附着在 ack 消息中返回给处理器核 i,并将同步变量 synid 中处理器核 i 的 W-notice 清空.

本方案中对每个同步变量为每个处理器核分配了 W-notice. 如果只对每个同步变量分配一个 W-notice,则随着程序不断地执行临界区代码,W-notice将不断累积. 这时即使某一个处理器核执行了 open _scope,也不能清除 W-notice,因为其它处理器核中仍可能存在属于 W-notice 集合的旧数据. 当 W-notice 累积满而不能被及时清除时,将导致每次 open_scope 操作时,处理器核都要将本地 DCache 全部无效掉.

通过为每个处理器核分配单独的 W-notice,使得同步管理器可以在处理 open _scope 后及时清除

W-notice,既避免了 W-notice 的不断累积,也不会使其它处理器核丢失一致性信息.在存储开销对比一节,我们将看到 W-notice 的存储空间开销是适度的.

对于栅障操作,各个处理器核的 W-notice 是参与栅障操作的所有其它处理器核合并的结果.当各处理器核的 W-set 较大时,会导致 W-notice 满.对于这种情况,DCache 相对于程序的数据集较小,在同步点进行一次无效掉全部 DCache 并不会对程序的性能带来明显影响.此外,在一些利用栅障进行同步的程序中,对于某一执行步骤,只有一部分处理器核执行程序并产生 W-set,而另外一部分处理器核只在同步点进行等待而并不产生 W-set,这时并不会造成 W-notice 满.典型的程序是 SPLASH-2 中的 LU 分解程序,当处理主对角线数据块时,只有负责该数据块的处理器核产生 W-set,其它处理器核并不产生 W-set.

3.6 支持嵌套的同步操作

支持同步操作的嵌套需要解决两个关键问题:(1)当临界区嵌套的深度大于片上支持的硬件同步变量个数时如何处理?(2)当临界区嵌套时,不同深度的一致性信息 W-set 如何继承与合并?

本方案中,每个节点的处理器核访存流水线中支持 2 个 W-set,当同步操作嵌套深度大于访存流水线中 W-set 个数时,通过采用类似 TLB 的机制,将外层同步变量替换到线程上下文中,并分配空闲出的 W-set 给内层同步变量使用.本方案中,每个节点的同步管理器支持 2 个同步变量,片上共支持 $2\times N_{synmanager}$ 个同步变量.对于 SPLASH-2 应用程序集,片上支持的硬件同步变量可以满足需求.

当发生同步变量嵌套时,在内层临界区的开始需要根据支持何种存储模型确定是否需要继承外层的 W-set.支持域一致性模型时,当执行内层 open_scope 指令时,初始化内层同步变量的 W-set 为空,当执行 close_scope 指令时,将本层的 W-set 合并到外层 W-set.支持释放一致性模型时,当执行内层 open_scope 指令时,内层 W-set 继承外层 W-set,当执行 close_scope 指令时,将本层的 W-set 合并到外层 W-set.

3.7 实现域一致性存储模型

本节介绍以硬件结构支持的同步协议实现域一致性存储模型时对访存操作和同步操作的完成次序的限制.要实现域一致性存储模型,必须保证:

条件 SYN_SCC_1:在处理器核 i 执行 open_scope synid 操作前,所有已执行的相对于同步变量 synid 的访存操作必须相对于处理器核 i 执行完;

条件 SYN_SCC_2:在处理器核 i 执行访存操作之前,所有程序序(program order)之前的 open_scope synid 都已经完成.

通过 2 个步骤满足条件 SYN_SCC_1,(1)当处理器核开始执行 close_scope 操作执行时,在临界区内的写操作在存储层次中最高一级的共享层次执行完.如图 1 所示,在存储层次中,DCache 是私有的,L2 是最高一级的共享存储层次.需要在 close_scope 操作执行前将 DCache 中状态为 DIRTY 的高速缓存行写回到 L2.进行 DCache 的写回操作时,可以从 W-set 中提取出临界区内写过哪些高速缓存行,而不必遍历整个 DCache.当从 DCache 中将 DIRTY 的高速缓存行写回后,此时临界区内的 store 操作相对于其它处理器核尚未完成,因为其它处理器核的 DCache 中仍可能存在旧数据.(2)当处理器核执行 open_scope 操作从同步管理器接收到 ack 消息时,处理器核从附着在 ack 消息上的 W-notice 中提取出要无效掉的高速缓存行,从本地 DCache 中将这些高速缓存行无效掉.这时,close_scope 操作之前的 store 操作相对于当前执行 open_scope 操作的处理器核完成.

对于条件 SYN_SCC_2,可以通过在执行 open_scope 操作时阻塞访存指令的执行实现.4.1 节提出了一种推测执行 open_scope 操作后的访存操作的方法.

通过改变 W-set 的作用范围,利用本方案中的机制还可以实现释放一致性存储模型和单项一致性存储模型等存储一致性模型.由于篇幅限制,在本文中不做详细介绍.

4 讨 论

本节讨论实现本方案的相关问题.4.1 节讨论如何在乱序执行处理器核中的推测执行临界区内访存操作;4.2 节讨论支持域一致性存储模型的系统的编程性问题;4.3 节讨论了假共享问题;4.4 节讨论了线程迁移问题.

4.1 乱序执行处理器核中推测执行临界区内的访存操作

按照域一致性存储模型的定义,当 open_scope 操作完成之前,程序序(program order)在其后的访存操作不能开始执行.如果按照定义来实现,由于限

制了 open_scope 之后的访存操作开始执行的时机,无法利用乱序执行的能力,限制了访存操作的性能.本方案中,推测 open_scope 操作之后的 load 操作执行是出于以下 3 个动机:(1) open_scope 操作需要访问同步管理器,延迟较长;(2) 临界区内的 load 操作有可能并不访问之前执行 close_scope 操作的处理器核产生的新值,这样即使 load 操作推测执行,也不会取得错误的数据;(3) 执行 open_scope 操作执行时,之前的 close_scope 可能已经完成,新产生的数据已经更新到共享的存储层次中,这样即使推测执行 load 操作,也可以取得新值.

文献[14]中提出了在目录协议中通过对 store 操作预取和对 load 操作推测执行以提高访存模型具体实现的性能.本方案中借鉴了文献[14]中的思想,即在 open_scope 操作尚未完成之前,其后 load 操作可以开始推测执行,load 操作的结果可以写回,供依赖其的后续操作继续执行.由于可能发生一致性冲突,即提前执行的 load 操作读取了旧值,所以必须要有机制来检测这种情况.在文献[14]中,这种检测是利用目录协议中的 intervention 消息实现的.本方案中,store 操作不产生 intervention 消息,一致性冲突检测是通过检验推测执行的 load 操作的访存地址是否属于 open_scope 操作返回的 W-notice来实现的.当推测执行的 load 操作的访存地址属于 W-notice 时,则发生一致性冲突,这时需要重新执行推测执行的 load 操作以及其后续操作.对于 store 操作的预取回填 DCache,同样利用 W-notice检测是否发生一致性冲突.

4.2　域一致性存储模型的编程性问题

域一致性模型对访存操作完成次序的要求弱于顺序一致性模型和释放一致性模型,所以针对后两者编写的并行程序在实现域一致性模型的系统上可能不能正确执行,如图 4 所示.

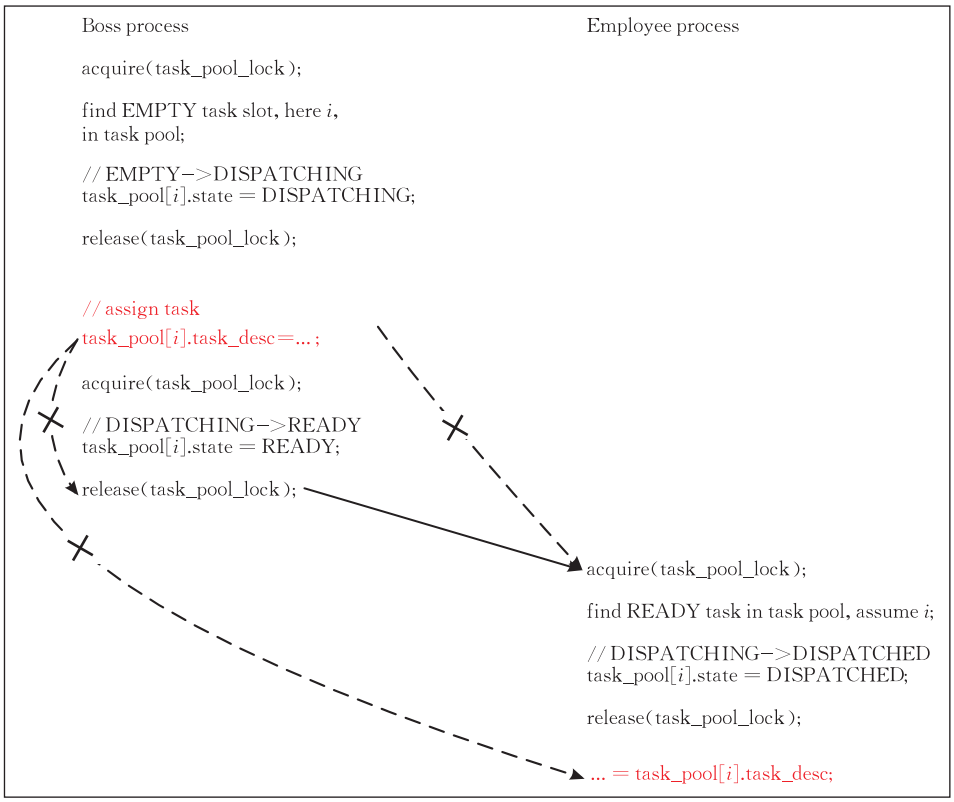
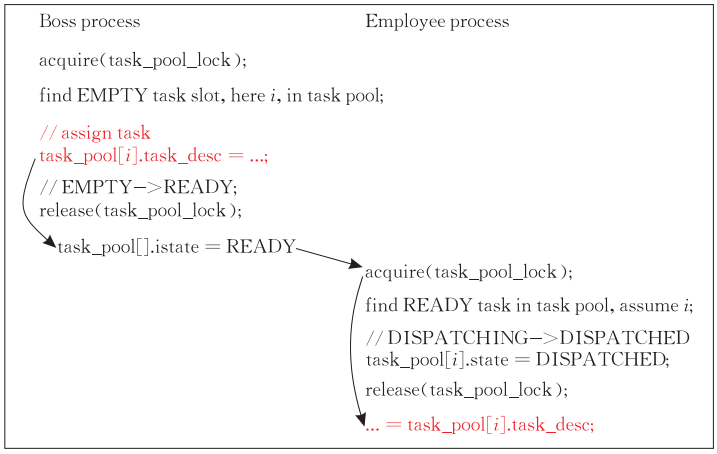


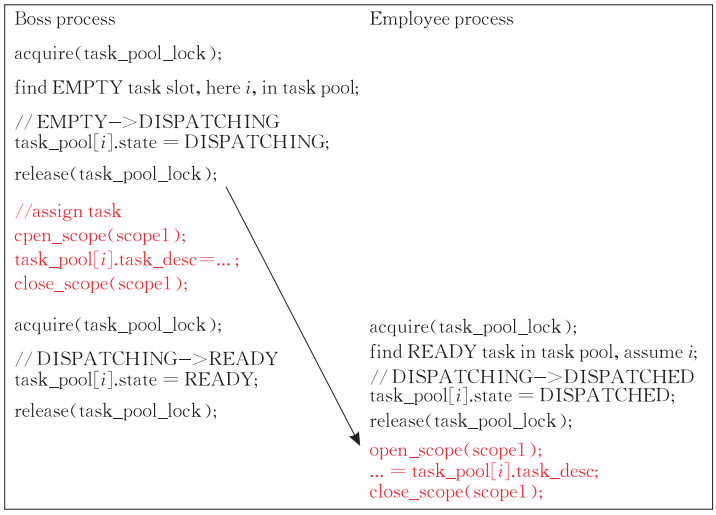
图 4　任务派发程序(虚线表示不存在顺序关系,实线表示存在顺序关系)

图 4 中程序示例了任务派发的过程,此处假设 boss 线程先于 employee 线程执行.程序中 boss 线程从任务池中取得空闲的任务描述符,对于任务描述符状态的修改在临界区内完成,而对于任务描述符内容的设置未受临界区的保护.以上程序在实现顺序一致性的系统可以正确执行;而在支持域一致性存储模型系统中,由于对任务描述符的访问不在由同步操作界定的一致性区间内,所以 employee 线程读取任务描述符的内容时,系统并不能保证 boss 线程更新任务描述符的写操作已经完成,在这种情况下 employee 线程可能得不到 boss 线程新写的值.

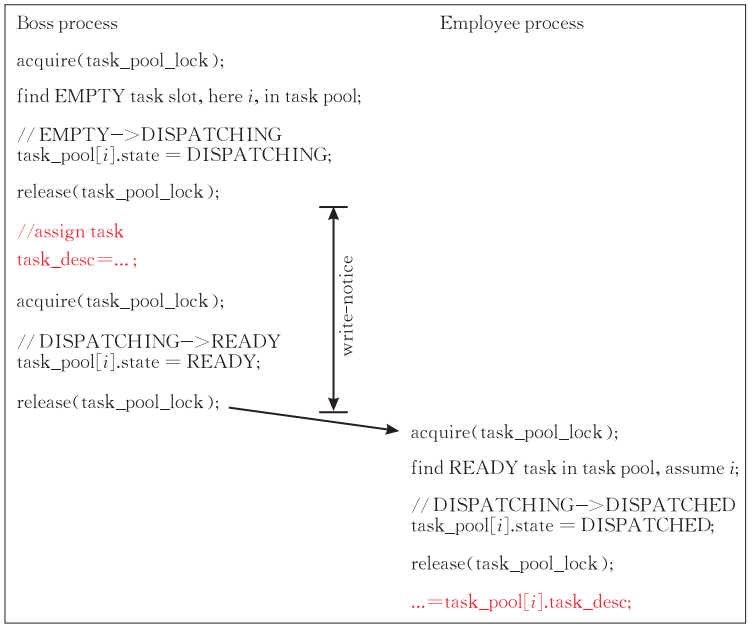
针对上面例子中的程序可以有 3 种解决方法：(1)硬件结构方法，即实现释放一致性模型，见图 5(a). 这种方法在两个 release 操作之间记录 W-set. SPLASH-2测试集中的程序主要以栅障操作进行



(a) 硬件方法



(b) 软件方法



(c) 软硬结合方法

图 5 编程性讨论

同步,对这种程序实现释放一致性模型和域一致性模型对应用程序的性能影响不大.对于以锁同步为主程序,在两个 release 操作之间记录 W-set,由于记录范围过大,会导致比较多的误识.(2)纯软件方法,即加大 acquire/release 的范围,如图 5(b)所示.这样会导致互斥的临界区增大,有损于系统性能.(3)软硬件结合的方法,见图 5(c).在上面的程序中,对于 task 描述符的操作通过 state 字段已经实现了互斥.但是在我们的方案中,一致性需求没有得到满足,当消费者程序需要读取 task 描述符的内容时,不能保证读取到新值.我们用 open_scope/close_scope指令定义一致性区间来保证一致性.如图 5(c)所示,通过 open_scope/close_scope 指令界定一致性区间,当消费者程序读取 task 描述符时,可以读取到新值.

域一致性存储模型的编程性问题主要体现在移植已有的面向顺序一致性存储模型等访存次序要求更为严格的存储模型上.由于域一致性存储模型本身符合用锁、栅障等常见同步操作实现的共享存储并行程序的编程模型,直接为实现域一致性存储模型的系统编写并行程序并不会有明显的编程性问题.

4.3　假共享问题

由于不使用目录,当执行 store 操作时并不无效掉在其它处理器核中同一高速缓存行的备份,所以一个高速缓存行可以同时在两个不同的处理器核中同时被改写.对于没有数据竞争(data race)的程序,如果同一高速缓存行的不同部分分别被不同处理器核上运行的线程改写,这种现象称为假共享.存在假共享的情况下,不能以整个高速缓存行为单位写回数据,否则会造成后写回的高速缓存行中的旧数据覆盖之前写回的新数据.直观的解决办法是在每个高速缓存行上扩展一个位向量,用于标识高速缓存行中哪些字段被改写.在写回时,只更新位向量标识出的数据.如果对每个字节用 1 位表示其是否被改写,则 DCache 需要增加相对于数据 1/8 的存储开销.本文的方法是每 4 个字节用 1 位表示其是否被改写过,对于写宽度是 4 字节 store 指令(SW)的写策略是 write back 的,而对于写宽度小于 4 字节的 store 指令(SH 和 SB)的写策略是 write through 的.这样增加的存储开销只有 1/32.实际的程序中,通常的对数据是以 4 字节(或以上)宽度进行读写,所以大多数 store 操作是 write back 的,而少量的 write through 并不会对性能产生明显影响.

4.4　线程迁移

要支持线程迁移,必须保证:(1)当发生线程切换时,用当前的 W-set 将 DCache 中的 DIRTY 的高速缓存行写回到 L2(如果可以确定线程下次仍被换入到这个处理器核,则可以不写回 DCache 中的数据);(2)将当前的 W-set 保存到线程的上下文.

当线程在执行 open_scope 操作而尚未接收到同步管理器发送的 ack 消息时如果发生线程迁移,这时:(1)要保证同步管理器将 ack 消息发送到新迁移到的处理器核;(2)当处理器核接收针对被换出的线程的 ack 消息时要能保证被换出的线程能够在换入后继续执行,以避免死锁情况.在实际的实现中,这些操作在操作系统中的进程管理模块中完成.

5　逻辑复杂度和存储开销上与目录协议的比较

要直接对比本文方案与目录协议的逻辑复杂度和存储开销,需要完全实现两种方案.本文对逻辑复杂度、存储开销给出说明性的对比.

虽然目录协议的原理是直观的,但是实际的实现中,由于以下的两方面原因导致其实现非常复杂:(1)维护目录本身的一致性,即目录信息与高速缓存块的实际状态不一致的情况.在引言中举出的例子即属于此类.在实际的实现中需要处理很多此类的边界情况.在基于同步的协议中,由于避免了目录,由此也避免了维护目录本身一致性的复杂度.同步管理器中的锁和栅障在逻辑上要比目录简单很多.(2)实现某种存储一致性模型带来的复杂度.在实现某一存储模型时,一方面必须考虑不同处理器核中存在冲突的访存操作之间的执行次序,另一方面还必须考虑在乱序执行的处理器核中的推测执行的访存操作是否违反一致性依赖.在目录协议中,这些是通过对 intervention 消息的处理来控制的.当接收到从目录发送的 intervention 消息时,处理器核的访存队列(LSQ)模块决定存在冲突的访存操作是否应该被无效掉,并且决定程序序在冲突访存之后而先于其执行的访存操作是否应该被无效掉.这些控制逻辑通常十分复杂,设计、实现和验证需要相当的时间.在基于同步的协议中,存储模型的实现逻辑集中于同步管理器模块,而不是像目录协议中由目录控制器(通常与 L2 控制器集成)与 LSQ 模块紧密交互实现,使得其在逻辑复杂度上相对较低.在乱序执行的处理器核中实现基于同步的协议时,在 LSQ 模块同样需要进行违反一致性依赖的检测,我们方案中用 bloom-filter 进行检测的方法与目录协议中用 intervention 消息进行检测的方法在逻辑复杂度上类似.

在存储空间开销方面，不同的配置下对比有所不同. 以下我们只考虑目录协议中的目录和基于同步的协议中 bloom-filter 的存储空间开销. 在目录协议中，我们假设采用 full bit vector 目录(压缩目录的方案可以降低存储空间开销，但是加剧了目录协议的逻辑复杂度，并且影响了性能). 在基于同步的协议中，每把锁需要片上处理器核数目个 bloom-filter. 以 64 核片上多核处理器为例，假设每个处理器核节点配置 1MB 的 L2，L2 缓存行大小为 64B，对于基于目录的协议，每个节点的目录所需存储空间共 1MB/64B×64Bit＝1MBit. 假设 bloom-filter 的宽度是 2048 位，每个处理器核节点支持两把硬件锁，共需 2048Bit×64×2＝256KBit，是目录协议的 1/4. 以下列出了不同配置下目录协议与基于同步协议的存储开销对比.

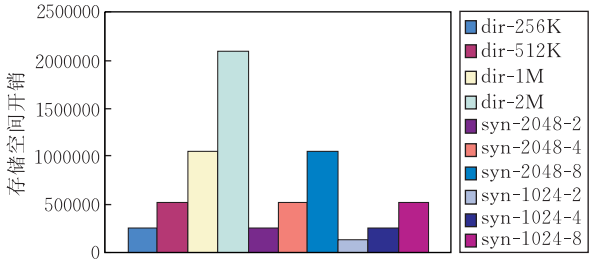


图 6 目录协议与同步协议存储开销对比

6 性能评估

我们在 ss-godson3 模拟器[15] 中实现了同步协议，并实现了域一致性存储模型，其中处理器核主要参数见表 4.

表 4 处理器核主要参数

Core			L2		
Fetch, issue, commit width	INT, FP, AGU	ROQ, BRQ, LSQ, MISSQ	Size	Line size	Associativity
4,4,4	2,2,1	64,8,24,8	1MB	32B	4-way
DCache				**NOC**	
Size	Line size	Associativity	W-set	Pipeline	Virtual Channel
64KB	32B	4-way	2KBit	routing, arbitration, traverse	REQ, INVN(dir), WTBK, RESP

为了对比性能，采用目录协议实现了处理器一致性存储模型(processor consistency). 对目录协议的模拟与真实的硬件逻辑十分接近[4]，并且采用了真实处理器中较为先进的技术实现了处理器一致性存储模型. 评估程序采用了 SPLASH-2 中的 LU、FFT 和 RADIX 程序，输入数据集采用标准大小.

图 7 和图 8 给出了同步协议的加速比特征和与目录协议的对比. 同步协议的加速比误识率定义如下：$|S_{W\text{-falsepositive}}|/|S_w|\times 100\%$，其中 S_W 是临界区内改写过的 cache line 地址的集合；$S_{W\text{-falsepositive}}$ 是误识的地址集合，定义为 $S_{W\text{-falsepositive}}=\{a\mid a\in S_{W\text{-bloomfilter}}$ 且 $a\notin S_W\}$，$S_{W\text{-bloomfilter}}$ 是以 bloom-filter 表示的临界区内改写过的 cache line 地址集合，即本文中的 W-set 和 W-notice. 图 9 给出了 Bloom-filter 的误识率特征(宽度为 2048-bit). 在所评估的程序中，误识率范围在 0.859%～3.14%，误识率较低说明 bloom-filter 在方案中的有效性. 对 Bloom-filter 宽度设计空间的探索是本文的后续工作之一.

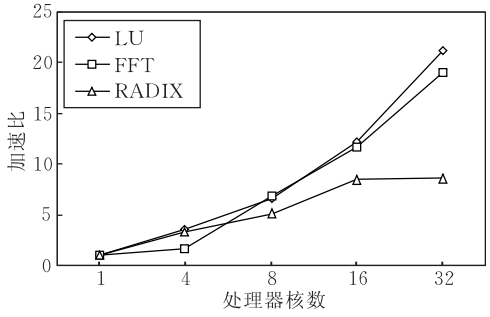


图 7 同步协议加速比特征

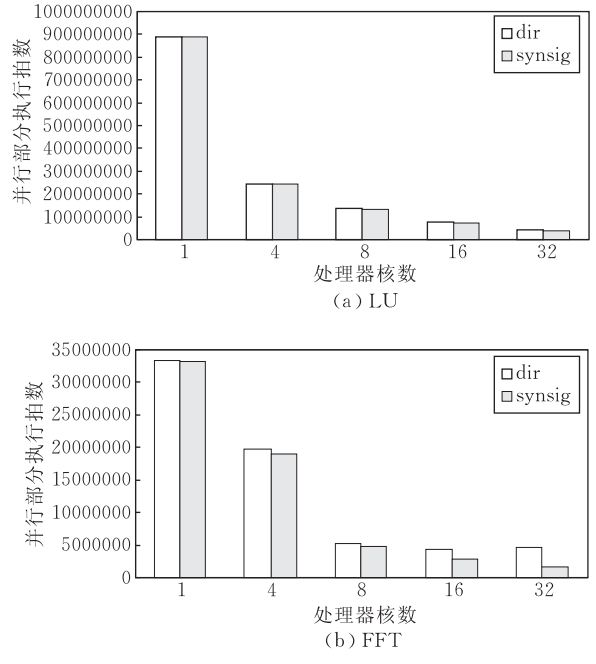


图 8 性能对比

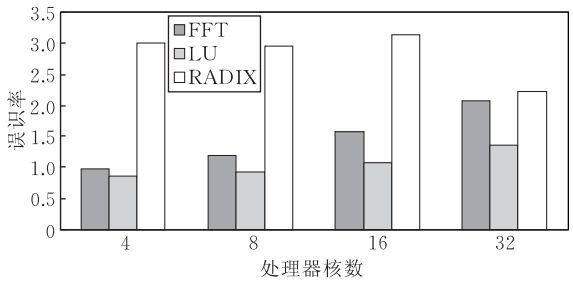


图 9　误识率特征

7　总　结

本文研究了在众核处理器中支持高速缓存一致性的方法,提出了硬件结构支持的基于同步的高速缓存一致性协议,并给出了利用该协议的片上多核处理器设计.基于同步的协议的核心思想是在并行程序的同步点而不是以高速缓存行为粒度支持高速缓存一致性;并且利用 bloom filter 高效地表示了一致性信息 W-set.基于同步的协议在设计、实现及验证复杂度上明显优于目录协议;在存储空间开销上,在未对两者进行优化时,同步协议中的 bloom filter 所需要的存储空间低于目录位向量所需空间,且 bloom filter 空间开销不随片上集成 L2 的增大而增大.在性能上,同步协议以临界区为粒度维护一致性,与目录协议以高速缓存行为粒度维护一致性相比,降低了一致性相关的片上网络消息量,从而性能上优于目录协议.同步协议相对于目录协议的优势在实际的片上多核处理器设计实现中有着积极的意义,一方面它使得芯片的设计、验证时间降低,另一方面协议所需较低的存储空间可以降低芯片成本或者利用节省的空间实现计算功能.

参　考　文　献

[1] Jeffrey K, David O, Mark H, John H, Richard S, Kourosh G, John C, David N, Joel B, Mark H, Anoop G, Mendel. The stanford FLASH multiprocessor//Proceedings of the 21st Annual ACM/IEEE International Symposium on Computer Architecture. Chicago, Illinois, USA, 1994: 302-313

[2] Dennis A, Steve S, David J L. So many states, so little time: verifying memory coherence in the Cray X1//Proceedings of the 17th International Parallel and Distributed Processing Symposium. Nice, France, 2003: 422-426

[3] Chaudhuri M, Heinrich M. SMTp: An architecture for next-generation scalable multi-threading//Proceedings of the 31st Annual ACM/IEEE International Symposium on Computer Architecture. Munchen, Germany, 2004: 124-135

[4] Daniel L, James L, Kourosh G, Anoop G, John H. The directory-based cache coherence protocol for the DASH multiprocessor//Proceedings of the 17th Annual ACM/IEEE International Symposium on Computer Architecture. Seattle, Washington, USA, 1990: 148-159

[5] Wentzlaff D, Griffin P, Hoffmann H, Bao L, Edwards B, Ramey C, Mattina M, Miao C, Brown J F, Agarwal A. On-chip interconnection architecture of the tile processor. IEEE Micro, 2007, 27(5)

[6] Kourosh G, Daniel L, James L, Phillip G, Anoop G, John H. Memory consistency and event ordering in scalable shared-memory multiprocessors//Proceedings of the 17th Annual ACM/IEEE International Symposium on Computer Architecture. Seattle, Washington, USA, 1990: 15-26

[7] Hu Wei-Wu. Shared Memory Architecture. Beijing: Higher Education Press, 2001(in Chinese)
(胡伟武. 共享存储系统结构. 北京: 高等教育出版社, 2001)

[8] Liviu I, Jaswinder P S, Kai Li. Scope consistency: A bridge between release consistency and entry consistency//Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures. Padua, Italy, 1996: 277-287

[9] Christian F, Marcelo C. An OS-based alternative to full hardware coherence on tiled CMPs//Proceedings of the 14th IEEE International Symposium on High Performance Computer Architecture. Salt Lake City, UT, USA, 2008

[10] Hakan Z, Zoran R, Martin K, Erik H. TMA: A trap-based memory architecture//Proceedings of the 20th Annual ACM International Conference on Supercomputing. Cairns, Queensland, Australia, 2006: 259-268

[11] Luis C, James T, Calin C, Josep T. Bulk disambiguation of speculative threads in multiprocessors//Proceedings of the 33rd Annual ACM/IEEE International Symposium on Computer Architecture. Boston, MA, USA, 2006: 227-238

[12] Luis C, James T, Pablo M, Josep T. BulkSC: Bulk enforcement of sequential consistency//Proceedings of the 34th Annual ACM/IEEE International Symposium on Computer Architecture. San Diego, California, USA, 2007: 278-289

[13] Burton H B. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970, 13(7): 422-426

[14] Kourosh G, Anoop G, John H. Two techniques to enhance the performance of memory consistency models//Proceedings of the 1991 International Conference on Parallel Processing. Austin, Texas, USA, 1991: 1355-1364

[15] Huang Kun, Ma Ke, Zeng Hong-Bo, Zhang Ge, Zhang Long-Bing. A user-level simulator for tiled chip multiprocessor. Journal of Software, 2008, 19(4): 1069-1080(in Chinese)
(黄琨, 马可, 曾洪博, 张戈, 章隆兵. 一种分片式多核处理器的用户级模拟器. 软件学报, 2008, 19(4): 1069-1080)

HUANG He, born in 1977, Ph. D. candidate. His main research interests focus on processor microarchitecture, many-core architecture, operating system and VLSI design.

LIU Lei, born in 1981, Ph. D. candidate. His main research interests focus on processor microarchitecture and low-power IC design.

SONG Feng-Long, born in 1980, Ph. D. candidate. His main research interests focus on on-chip memory systems.

MA Xiao-Yu, born in 1984, Ph. D. candidate. His main research interests focus on chip-multiprocessor architecture.

Background

The cache coherence protocol is a first-order design consideration in shared memory multicore design. Although distributed directory-based protocols have been proven with fairly scalability, reaching up to hundreds of processors in multiprocessor CCNUMA DSM system, they have many weaknesses preventing it from adapting in many-core architectures. For examples, directory protocols are very difficult to completely debug and verify due to subtle corner cases and tremendous state transitions. There are limited research results related to cache coherence issue for many-core architectures. In this paper, a new hardware cache coherence scheme, architecture supported synchronization-based cache coherence protocol is proposed, which achieves similar performance in cost-effective way compared to a directory-based protocol that requires large amount of hardware resources and huge design verification effort.

The project is supported by the National Basic Research Program (973 Program) of China under grant No. 2005CB321600 and the National Natural Science Foundation of China under grant No. 60736012.