doi:10.3772/j.issn.1002-0470.2013.11.002

用于多核同步优化的 cache 一致性协议设计①

陈李维②[*] [**] [***] [****] 张广飞[*] [**] [***] [****] 汪文祥[*] [**] [****] 王焕东[*] [**] [****] 李 玲[*] [**] [****]

([*]中国科学院计算机体系结构国家重点实验室 北京 100190)
([**]中国科学院计算技术研究所 北京 100190)
([***]中国科学院大学 北京 100049)
([****]龙芯中科技术有限公司 北京 100190)

摘 要 通过对多核同步过程中的访存行为进行分析,提出了一种识别同步类型的方法,并设计了一种实现同步优化的新的 cache 一致性协议。该协议增加了一个用于记录同步信息的 cache 状态,通过阻塞的方式可以让多个处理器核串行地完成同步操作,保证同步操作中原子指令能够顺利执行成功,从而大大减少由多核同步冲突引发的访存请求数量,将多核同步过程中的访存行为优化到了几乎最好的情况。实验结果表明,通过同步优化,这个新的 cache 一致性协议能够使多核同步的性能提升到接近最理想的结果。实验表明,相比传统的 cache 一致性协议,实验中采用的几个标准多核性能测试程序优化后的同步性能提升了 1 倍,而并行程序整体运行时间降低 25%。

关键词 同步,栅障,锁,cache 一致性协议,片上多核处理器(CMPs)

0 引 言

随着芯片中处理器核数量的增多和处理能力的增强,程序中可以并行的部分执行速度越来越快,然而串行部分,如多核同步操作,效率不仅没有随着核数量的增加而得到提高,反而因为引入了大量访存冲突而使系统性能有所降低,因此并行程序中同步操作成为片上多核处理器(chip multiprocessors, CMPs)性能提升的一个瓶颈[1]。本文针对这个问题进行了研究。为了提高同步执行效率,提出了一种针对同步优化的新的 cache 一致性协议,实验表明,这个新的协议可显著提高多核处理器的性能。

1 相关研究

Rajwar 等[2]于 2000 年提出了提高同步效率的两个关键步骤:识别出正在执行的同步类型和优化同步性能。

(1)识别同步类型。同步操作都需要使用特殊的原子指令,因此识别出同步操作比较简单。但是识别出同步操作的类型则比较困难。并行程序中的同步操作主要有两种类型,栅障(barrier)和锁(lock)[3]。栅障确保了多核处理器程序执行的同步性,只有当所有处理器都到达栅障时处理器才能继续往下执行。锁确保了多核处理器程序执行的互斥性,只有获得锁的处理器才能执行特定的一段程序。相关研究工作[2]主要通过查表的方式来预测目前同步的类型。但是增加一个记录同步类型的表增加了硬件设计的复杂度,而且查表预测的精度和表的大小以及应用程序类型有关,对于某些应用程序来说同步类型的预测精度可能不够高。本文通过对同步处理的分析发现,在原子指令执行成功之后,不同类型的同步会执行不同的访存操作。其中栅障会发出一个 load 指令,读取并判断是否到达栅障边界,而锁会发出一个存储(store)指令来释放锁。因此,观察原子指令之后的访存操作可以判断出同步的类型。

① 国家"核高基"科技重大专项课题(2009ZX01028-002-003, 2009ZX01029-001-003, 2010ZX01036-001-002),国家自然科学基金(60921002,61003064,61050002,61070025,61100163,61133004,61173001,61232009)和 863 计划(2012AA010901,2012AA011002,2012AA012202,2013AA014301)资助项目。

② 男,1986 年生,博士生;研究方向:计算机系统结构,多媒体优化;联系人,E-mail:chenliwei@ict.ac.cn

(收稿日期:2012-11-23)

(2)优化同步性能。为了保证并行程序的正确性,同步操作必须串行地执行。此外,对于基于写无效的高速缓存(cache)一致性协议,当多个处理器核同时开始进行同步时,会互相无效对方的 cache 块,从而产生很多访存冲突,导致片上网络中数据传输的拥塞。基于队列的串行化方法是优化同步性能的一种常用的方法[4],该方法规定了处理器核执行同步的顺序,能有效避免访存冲突。但是目前大部分基于队列的同步优化方法[2,3,5,6]都需要复杂的软硬件支持或者特殊的指令。此外,对同步进行优化还有其他方法,如 RW-lock[7]和 trylock[8]等。RW-lock 允许多个处理器同时读取一个锁,提高了同步操作的并行性。而当一个处理器在排队等待获得同步执行权限时,如果等待时间过长,trylock 机制允许取消等待转而执行其他不相关程序,过一段时间之后再重新尝试执行同步,避免了处理器长时间的空闲。

为了提高多核同步的效率,本文提出了一种识别同步类型的方法,并实现了一种针对同步优化的 cache 一致性协议。与传统 cache 一致性协议相比,该 cache 一致性协议只需要增加一种新的 cache 状态——加载关联独占状态(LLEXC),设计复杂度低,代价很小。这个新的 cache 状态用于记录当前同步执行的信息,能够维护多核同步执行的串行性。本文采用了加载关联/条件存储(load-linked/store-conditional,LL/SC)指令作为同步操作中的原子指令。LL/SC 是一种经典的原子指令,被多种常见的现代处理器指令集采用[9-11]。实验结果表明,在 16 核高速缓存一致性—非均匀存储访问(cache coherent non-uniform memory accessing,CC-NUMA)系统中,采用新的 cache 一致性协议,通用并行程序的性能提升了 25%。

2 同步操作分析

同步是一种特殊的访存操作,多个处理器核通过对一个共享内存地址读写来传递信息,主要开销就是访问不命中引发的访存开销[12]。首先,处理器核使用 load 指令(LL)读取一个共享内存地址的数据,然后使用 store 指令(SC)修改这个共享内存地址的值,这整个过程需要保证执行过程的原子性。如果有多个处理器核同时修改这个共享内存地址,一次最多只能有一个处理器核能够成功完成修改操作,其他处理器核需要重新进行修改操作。传统的 cache 一致性协议没有很好地针对同步的原子性进行优化,尤其是基于目录的写无效 cache 一致性协议。多个 load/store 指令可能使得同步的原子性被破坏,原子指令(LL/SC)执行失败,导致原子指令不断地重复执行。图 1 分别描述了两种不同同步操作(栅障和锁)的汇编指令的基本实现,下面分别详细介绍一下栅障和锁的实现过程。

```
C0:
    LLa 0, Barrier          # load, 读取内存
    ADDIU a 0, a 0, 0x1     # 加一
    SC a 0, Barrier         # 尝试存储, 检查原子性
    BEQ a 0, zero , C0      # 如果原子性被破坏(值为0), 重新执行
    NOP
C1:
    LI a 1, 0x10            # 设置栅障边界值(假设为0x10)
    LD a 0, Barrier         # load, 读取
    BNE a 0, a1, C1         # 如果不等于栅障边界值, 重新执行
    NOP
```

(a) 栅障(Barrier)的实现

```
C0:
    LLa 0, Lock             # load, 检查锁
    BNE a 0, zero , C0      # 如果锁已经被其他核获得(值为1), 重新执行
    NOP
    LI a 1, 0x1
    SC a 1, Lock            # 尝试获得锁, 将锁的值设为1, 检查原子性
    BEQ a 1, zero , C0      # 如果原子性被破坏, 重新执行
    NOP
    .
    .                       # 获得锁以后需要执行的一段特殊程序
    .
    SD zero , Lock          # 释放锁, 将锁的值设为0
```

(b) 锁(Lock)的实现

图 1 实现同步过程的汇编代码:(a)栅障的实现;(b)锁的实现

(1)栅障。处理器核首先使用 LL 指令读取栅障对应的共享数据,然后将这个数据加 1 之后再通过 SC 指令存回。如果在 SC 指令完成前发现其他处理器核已经修改对应的数据了,则 SC 指令执行失败并返回 0,修改后的数据也不会真正写入内存。如果 SC 执行失败,则需要重复执行 LL/SC 指令,直到成功为止。当 SC 执行成功之后,处理器接下来会使用普通 load 指令读取对应地址的值,如果发现该值已经到达边界值,则认为所有处理器已经到达栅障,可以继续往下执行了,否则,则认为还有一些处理器没有到达栅障,需要循环等待。

(2)锁。首先,处理器核通过 LL 指令读取锁对应的共享数据。如果锁已经被其他处理器核占用了(值为 1),则循环读取直到锁被释放。如果锁已经空闲(值为 0),则通过 SC 指令将对应数据修改为 1,获取锁。如果 SC 执行失败,则重复执行 LL/SC。如果 SC 执行成功,说明已经获得锁,则可以执行锁对应的特定共享程序。当对应程序执行完成,再通

过一个普通的 store 操作将锁释放。

栅障和锁的一个主要的区别在于原子指令(LL/SC)执行完成之后的访存操作。如果这个访存操作是 load 指令,则说明该同步操作是栅障。反之,如果这个访存操作是 store 指令,则说明该同步操作是锁。此外,对于锁操作来说,当一个处理器核执行完成 SC 操作以后,下一个处理器并不能继续执行 SC 操作,因为锁已经被前一个处理器获得。而对于栅障操作来说,当前一个处理器成功完成 SC 操作之后,下一个处理器可以马上开始执行 LL/SC 操作。

3 同步优化实现

首先,基于传统的写无效 cache 一致性协议描述多核同步的实现过程,并分析造成同步效率低下的原因。然后,针对多核同步优化,给出一个最理想情况下的实现过程。最后,提出一个新的 cache 一致性协议,使得多核同步能够尽可能接近最理想的实现过程。

3.1 多核同步性能分析

为了简化同步过程分析,假设在一个传统的基于目录写无效 cache 一致性协议中,每一个 cache 块只有三种最基本的状态:空闲状态(IDLE)、共享状态(SHD)和独占状态(EXC)。IDLE 表示当前 cache 块是无效的,对其进行任何读写操作都会引发缓存缺失(cache miss)。SHD 表明当前 cache 块可能被多个处理器核共享,只能读取,不能写入,对其写入也会引发 cache miss。EXC 表明对应 cache 块被当前处理器核独占,该处理器核可以任意地读写这个 cache 块。此外,假设在这个片上多核系统中,每一个处理器核都有一个私有的一级 cache,而二级 cache 由所有处理器核所共享的。上述 cache 一致性协议中的相关信息就保存在共享二级 cache 的目录中。

图 2 显示了在上述片上多核系统中,一个典型的同步处理流程。两个处理器核 P0 和 P1 同时使用 LL 指令对同一共享地址进行访问,所以分别发出了两个读共享请求给共享二级 cache。当获得读数据响应以后,两个处理器核再分别执行 SC 指令,分别发出读独占请求。假设 P0 的读独占请求先于 P1 到达,则二级 cache 根据目录信息向所有包含对应 cache 备份的处理器核发出无效请求。当二级 cache 收到所有的无效响应以后,将读独占数据发给 P0。因此,P1 的 LL/SC 操作的原子性被破坏,需要重新执行 LL/SC 原子操作。因为图 2 的同步过程是锁操作,所以 P0 在 SC 指令成功以后获得锁,开始执行锁对应的程序段(critical section),而 P1 则再一次试图获取锁。P1 第二次执行 LL 指令,发出读共享请求。二级 cache 收到 P1 的请求后,根据目录信息向 P0 发出写回请求。然后,P0 将相应数据写回,同时将 cache 块状态从 EXC 改为 SHD。最后,二级 cache 将读共享数据发给 P1。但是,当 P1 读取到锁内容以后,发现锁已经被 P0 获取了,因此只能继续循环等待 P0 释放锁。当 P0 最终释放锁时,需要执行一个 store 指令,而写操作需要 EXC 的 cache 状态,因此 P0 需要再发出一个读独占请求,从而会再次无效 P1 中 cache 块的内容。可以发现,在 P0 获得锁之后,P1 再读取锁内容是没有意义的,因为 P1 无法获得锁,而且 P1 中对应的 cache 块最终都会因为 P0 释放锁的 store 指令而被无效掉。

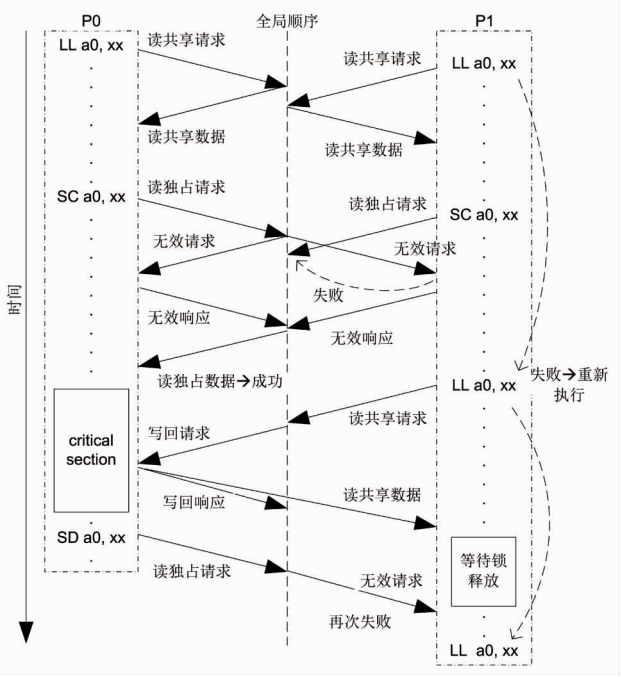

图 2 同步(锁)在传统系统中的实现过程

根据以上的分析,在传统的写无效 cache 一致性协议中,多核同步处理过程中有大量没有意义的访存操作,如 P1 在 P0 获取锁以后发出的 LL 指令引发了多个访存操作。此外,随着参与同步的处理器核数量的增加,同时开始执行同步操作的概率也在增加,因此,SC 执行失败的概率和不必要的访存操作数量也大大增加了,可能造成片上网络数据传输拥塞,导致严重的带宽资源浪费和大量无意义的功耗损失。

LL/SC 原子指令也带来了访存操作的浪费。首先 LL 指令引发一次读共享请求,获得相应的读共

享数据响应以后,接下来的 SC 指令会引发一次读独占请求,最后获得读独占数据响应,将对应 cache 块状态从 SHD 变为 EXC。因此在同步过程中,就算是一切顺利,LL/SC 也至少会产生两次访存操作。

3.2 理想的多核同步过程

为了提高多核同步的性能,最重要的就是减少多余的访存操作。根据对基于队列同步优化方法的分析和总结,图 3 描述了一个理想的同步实现过程。理想情况是指多核同步过程中,多个处理器核能串行地依次完成相应同步操作,所有的访存操作都是必须的,没有任何多余的操作,而且访存操作都是及时发出,没有任何不必要的延时(不考虑在片上网络上传输的延迟)。根据第 3.1 小节的分析,需要分别优化 LL/SC 本身带来的访存操作浪费和多处理器核竞争冲突导致 SC 指令执行失败引发的多余访存操作。

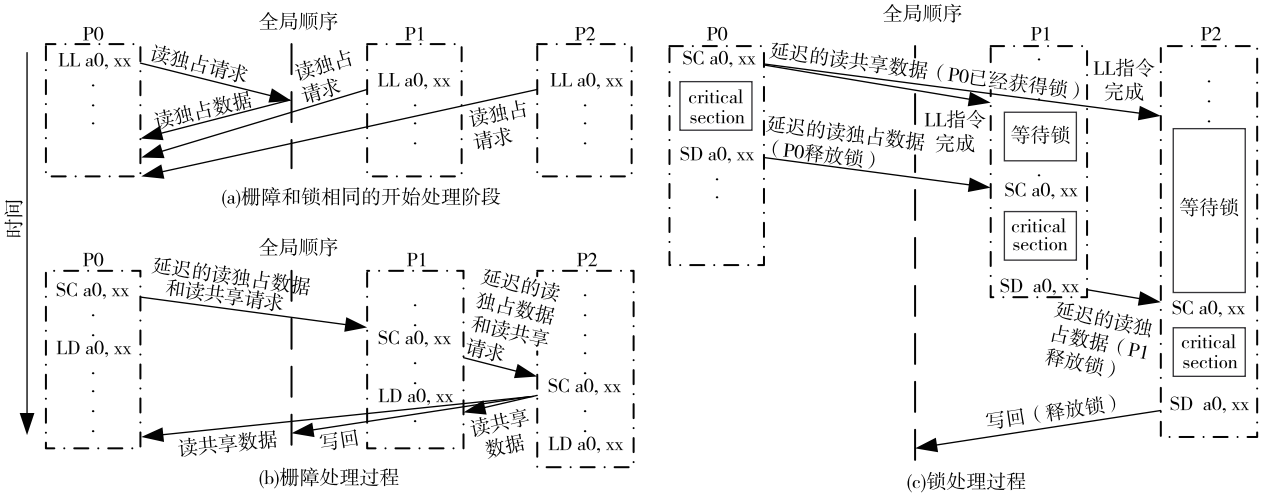

图 3 在理想情况下的同步实现:(a)栅障和锁相同的开始处理阶段;(b)栅障处理过程;(c)锁处理过程

在同步过程中,LL 指令和 SC 指令之间的间隔很小,往往只有 1 - 2 条计算指令,因此,直接让 LL 指令发出读独占请求,SC 指令就不需要再发读独占请求了,从而可以节省一个访存操作。如图 3(a)所示,三个处理器核 P0、P1 和 P2 分别执行 LL 指令并发出了读独占请求。

(1)假设该同步操作是一个栅障,如图 3(b)所示。P0 最先获得了读独占数据,而 P1 和 P2 的读独占请求也被转发给了 P0。为了避免 SC 执行失败导致的多余访存操作,P0 将 P1 和 P2 发出的读独占请求暂时阻塞,直到 P0 的 SC 指令执行成功。在理想情况下,P0 立即知道当前的同步操作是一个栅障,因此,当 SC 执行完成后,P0 将发送一个信息给下一个有需要的处理器核(P1),该信息包含读独占数据、一个共享的读请求(为了 SC 指令之后的 load 指令)和其他需要读独占数据的处理器核(如 P2)。和 P0 类似,P1 收到 P0 发出的信息以后,顺利完成 LL/SC 指令,然后向 P2 发出一个信息,包含读独占数据和读共享请求(P0 和 P1)。最后,P2 收到 P1 发出的信息,完成 LL/SC 原子操作。P2 发现没有处理器核需要读独占数据了,所以 P2 根据收到信息中包含的读共享请求分别将读共享数据发给 P0 和 P1,同时将该数据写回二级 cache。

(2)假设同步操作是一个锁,如图 3(c)所示。因为获得锁之后需要执行的程序段大小不是固定的,执行完成所需要的时间可能比较长,所以一直阻塞来自 P1 和 P2 的读独占请求导致相应的 LL 指令无法完成,从而阻塞 P1 和 P2 的整个流水线,可能会降低系统性能。因此,P0 完成 SC 指令获得锁以后,分别发送读共享数据给 P1 和 P2,让 P1 和 P2 的 LL 指令能够顺利提交。因为锁仍然被 P0 获得,P1 和 P2 实际上并不会修改锁的值。此外,P0 的 cache 块仍然是独占状态,而 P1 和 P2 的 cache 块是共享状态。当 P0 最终完成执行,通过 store 指令释放锁以后,P0 发送一个信息给 P1,该信息包含读独占数据和其他需要读独占数据的处理器核(如 P2),然后 P1 就可以获得锁并往下执行了。当 P1 执行完成后,P1 释放锁,并发送一个信息给 P2。最后,P2 完成所有操作,释放锁。P2 发现已经没有处理器核需要读独占数据了,因此将数据最终写回 L2 cache。

3.3 多核同步实际优化过程

为了让多核同步过程的实现能够接近第 3.2 小节所描述的理想情况,在传统的 cache 一致性协议基础上增加了一个用于维护同步信息的 cache 状

态,LLEXC。图 4 显示了在同步过程中 cache 块状态的相互转换。当处理器核执行 LL 指令发出读独占请求并收到相应的读独占数据响应之后,需要将对应的 cache 块状态改为 LLEXC。然后,SC 指令可以直接对 LLEXC 状态的 cache 块进行修改。假设这个同步操作是栅障,则处理器核会在 SC 指令之后马上执行一条 load 指令,然后 cache 块将状态转为 SHD,并将读独占数据发给下一个发出读独占请求的处理器核。假设这个同步操作是锁,则在执行完锁对应的程序段以后,处理器核会执行 store 指令释放锁,然后 cache 块将状态转为 IDLE,并将读独占数据发给下一个发出读独占请求的处理器核。

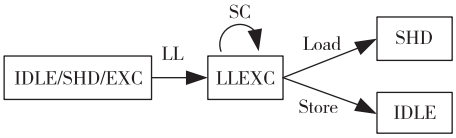

图 4 同步过程中 cache 块的状态转换

图 5 显示新的 cache 一致性协议的同步处理流程和理想情况下是比较接近的。图 5(a)和图 3(a)完全一样,三个处理器核 P0、P1 和 P2 分别根据 LL 指令发出了读独占请求。

假设该同步操作是一个栅障,如图 5(b)中所示。根据对同步过程的分析,需要等到 load 指令才能真正确定该同步是一个栅障,因此,与理想情况相比,实际同步过程有了一个延迟。但是,根据第 2 节的分析,load 指令和 SC 指令之间的间隔非常小,延迟也很小,因此,对性能的影响可以忽略不计。

图 5(c)展示了实际情况下锁处理的流程。因为在 SC 指令执行完成后,P0 并不知道目前的同步操作是哪一种类型,因此只能一直阻塞 P1 和 P2 发出的读独占请求。只有当 P0 完成锁对应的程序段以后,执行 store 指令释放锁时,才能真正确定当前的同步是一个锁操作。因此,与理想情况相比,P1 和 P2 的 LL 指令一直不能完成,直到 P0 释放锁。而根据第 2 节的分析,就算 P1 和 P2 的 LL 指令能够尽快完成,也只能循环执行,等待 P0 释放锁。因此,LL 指令被阻塞对实际的同步性能并没有影响,反而因为减少了通知 P1 和 P2 的两个读共享数据响应操作,节省了片上网络传输带宽,但是可能会对线程切换等系统运行造成一定的阻碍。最后,根据实验可以发现,这样的情况还是可以接受的。

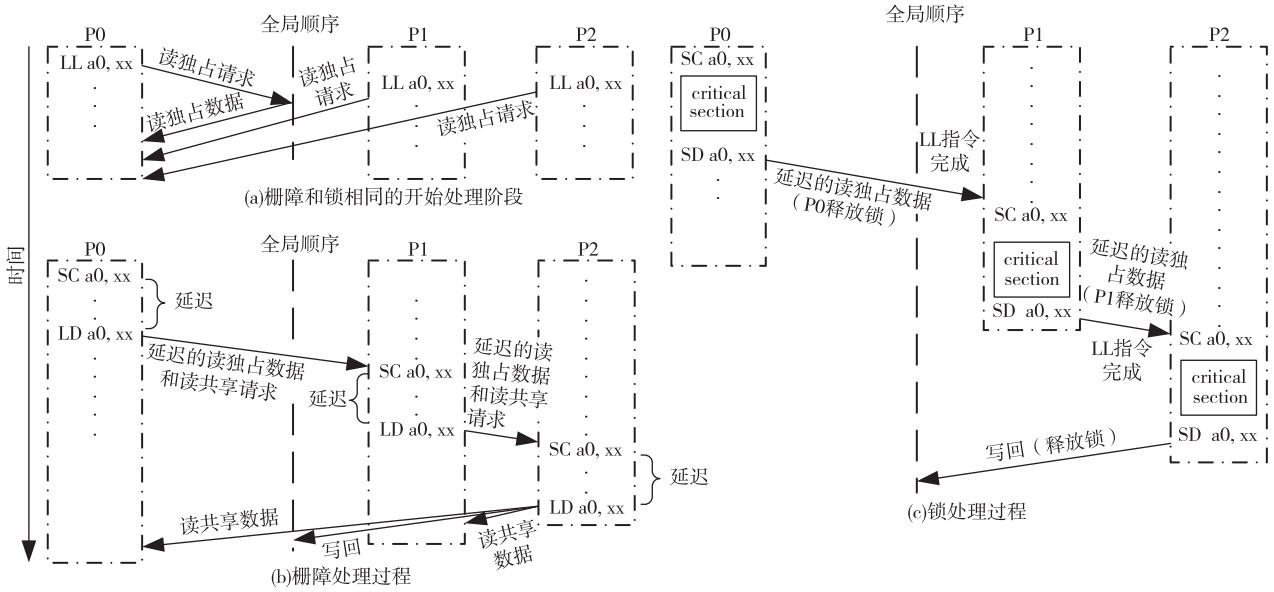

图 5 优化后的同步实现,(a)栅障和锁相同的开始处理阶段,(b)栅障处理过程,(c)锁处理过程

4 性能分析

4.1 实验环境

为评估上面提出的优化方法,本文选用一个典型的 16 核 CC-NUMA 系统,其中处理器核采用了 64 位超标量精简指令集计算机(RISC)结构,私有一级 cache 和共享分片式二级 cache。具体的系统参数如表 1 所示。整个系统是寄存器传送级(register transfer level,RTL)模拟器,采用 Verilog 硬件描述语言编写,运行在 EVE Zebu 高速仿真平台上。本文从常见的标准多核性能测试程序 SPLASH-2[13] 中选择了几个典型的测试程序用于同步性能测试,如表 2 所示。在 FMM 和 Ocean 中,大部分同步操作都是栅障,而在

Barnes 和 Radiosity 中,锁操作的执行时间要大于栅障。在 Water-ns 和 Volrend 中,栅障和锁的执行时间基本相同。

表 1　系统参数

参数	描　　述
CMPs	16 核,2D-Mesh
处理器核	4 发射 64 位超标量,乱序执行,1GHz
L1 Cache	64KB 4 路组相联 I-Cache 和 D-Cache
L2 Cache	1MB/核,4 路组相联
主存	4GB,访问延迟 256 拍

表 2　测试程序参数

测试程序	描　　述
FMM	16K particles
Ocean	258x258 ocean
Barnes	16K particles
Radiosity	test,-bf 0.1,-ae 1000.0-en 0.005
Water-Nsquared	512 molec.
Volrend	256x256x126 head

本文将整个并行程序执行时间分为 4 个部分:栅障执行时间(barrier)、锁执行时间(lock)、运算时间(compute)和访存时间(memory)。用于栅障和锁的指令执行时间分布是栅障执行时间和锁执行时间。访存时间是指除了用于同步操作以外的所有 load/store 指令的执行时间。运算时间是指除了同步操作和 load/store 指令以外所有指令的执行时间。

4.2　实验结果

图 6 显示了在不同实现下并行程序的执行时间。与原始的 cache 一致性协议相比,本文提出的方法将同步的执行时间减少了 1 倍以上,并将系统性能整体提升了 20% ~30%。此外,因为多核同步冲突引发的访存数量降低了不少,片上网络传输更加顺畅,所以普通访存操作的时间也降低了 10% 左右。与理想的系统相比,本文的方法只有约 1% 的性能降低。

基于队列的方法对于同步优化非常有效,特别是同步冲突很严重的情况。同步执行所占时间越大,说明同步冲突越严重,本文的优化方法效果也就越明显,并行程序性能提升更大。例如,在 FMM 中,栅障执行时间占整个程序执行时间的 40%,优化以后,栅障的执行时间减少了 3 倍以上。而在 Radiosity 中,栅障的执行时间只占整个程序执行时间的 10%,因此,栅障经过优化以后的执行时间只减少了约 40%。

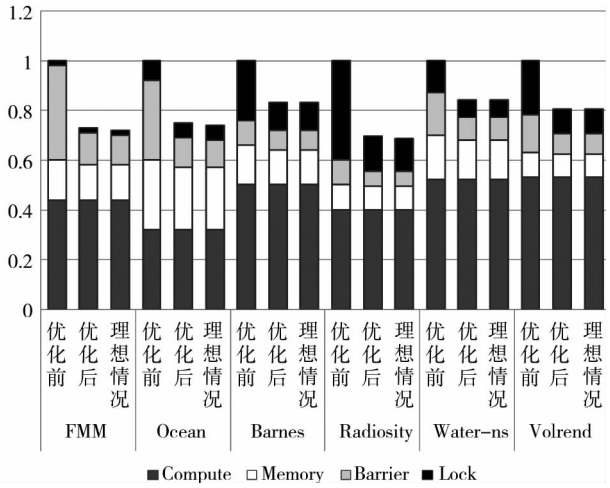

图 6　程序执行时间(根据优化前的执行时间进行归一化)

4　结　论

为了提高同步执行的效率,本文提出了一种新的 cache 一致性协议,保持了多个处理器核同步执行的串行化。此外,根据原子操作之后的 load/store 指令不同可以分辨出同步操作的类型,从而可以有针对性地进行优化。经过优化以后,同步在执行过程中没有任何多余的没有意义的访存操作,从而有效地提高了性能。实验结果表明,与原始的 cache 一致性协议相比,本文提出的方法将同步性能提升了 1 倍以上,将并行程序的整体性能提升了 25% 左右。此外,本文的方法实现代价非常小,只需要增加一个 cache 状态,不需要额外的特殊指令或者复杂的软硬件支持。本文主要针对同步引发的多余访存操作进行优化,除此之外还有一些其他的同步优化方法[14-16]。例如,因为同步只能串行的执行,性能比并行的程序要低,因此,使用多个处理器核中最快的处理器核来执行同步可以提高同步的处理速度[15]。这样的方法可以在本文的基础上进一步提高同步的执行效率。

参考文献

[1] Sahelices B, de Dios A, Ibáñez P, et al. Efficient handling of lock hand-off in DSM multiprocessors with buffering coherence controllers. *Journal of Computer Science and Technology*, 2012, 27: 75-91

[2] Rajwar R, Kägi, A, Goodman J. Improving the throughput of synchronization by insertion of delays. In: Proceedings of the 6th International Symposium on High-Performance

Computer Architecture, Touluse, France, 2000. 168-179
[3] Mellor-Crummey J M, Scott M L. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 1991, 9(1): 21-65
[4] Monchiero M, Palermo G, Silvano C, et al. Power/performance hardware optimization for synchronization intensive applications in MPSoCs. In: Proceedings of the Conference on Design, Automation and Test in Europe, Munich, Germany, 2006. 1-6
[5] Goodman J R, Vernon M K, Woest P J. Efficient synchronization primitives for large-scale cache-coherent multiprocessors. In: Proceedings of the 3rd International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, USA, 1989. 64-75
[6] Kägi A, Burger D, Goodman J R. Efficient synchronization: let them eat QOLB. In: Proceedings of the 24th Annual International Symposium on Computer Architecture, Denver, USA, 1997. 170-180
[7] Vallejo E, Beivide R, Cristal A, et al. Architectural support for fair reader-writer locking. In: Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture, Atlanta, USA, 2010. 275-286
[8] Bron A, Farchi E, Magid E, et al. Applications of synchronization coverage. In: Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Chicago, USA, 2005. 206-212
[9] Kane G. MIPS RISC Architecture. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988
[10] May C, Silha E, Simpson R, et al. The PowerPC Architecture: A Specification for A New Family of RISC Processors. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994
[11] Sites R L. Alpha AXP architecture. *ACM Communications*, 1993, 36(2): 33-44
[12] 蔡晔, 史岗. 基于锁的 Cache 一致性协议的硬件优化策略. 高技术通讯, 2009, 19(9): 933-938
[13] Woo S, Ohara M, Torrie E, et al. The SPLASH-2 programs: characterization and methodological considerations. In: Proceedings of the 22nd Annual International Symposium on Computer Architecture, S. Margherita Ligure, Italy, 1995. 24-36
[14] Rajwar R, Goodman J R. Speculative lock elision: enabling highly concurrent multithreaded execution. In: Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture, Austin, USA, 2001. 294-305
[15] Suleman M A, Mutlu O, Qureshi M K, et al. Accelerating critical section execution with asymmetric multi-core architectures. In: Proceedings of the 14th international conference on Architectural support for programming languages and operating systems, Washington, D C, USA, 2009. 253-264
[16] Zhu W, Sreehar V C, Hu Z, et al. Synchronization state buffer: supporting efficient fine-grain synchronization on many-core architectures. In: Proceedings of the 34th Annual International Symposium on Computer Architecture, San Diego, USA, 2007. 35-45

A cache coherence protocol for optimizing CMP synchronization

Chen Liwei[*,**,***,****], Zhang Guangfei[*,**,***,****], Wang Wenxiang[*,**,****],
Wang Huandong[*,**,****], Li Ling[*,**,****]
([*] Key Laboratory of Computer System and Architecture, Chinese Academy of Sciences, Beijing 100190)
([**] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)
([***] Graduate University of Chinese Academy of Sciences, Beijing 100049)
([****] Loongson Corporation, Beijing 100049)

Abstract

Based on the memory accessing analysis for the synchronization process of chip multiprocessors (CMPs), a method to recognize the type of synchronization was proposed, and a novel cache coherence protocol for optimization of the synchronization of CMPs was designed. The protocol adds a new cache state for synchronization information, and it can make CMPs realize the serial synchronization operation with the way of blocking to guarantee successful execution of atomic operations. Thus, the number of memory accesses caused by synchronization conflicts can be greatly reduced, and an almost perfect memory accessing process in synchronization can be achieved. The experimental results show that, with the proposed cache coherence protocol, the synchronization performance is almost perfect. Compared with the traditional cache coherence, the synchronization performance can be increared by 100%, and the whole execution time of parallel programs can be reduced by 25%.

Key words: synchronization, barrier, lock, cache coherence protocol, chip multiprocessors (CMPs)