高技术通讯 2009 年 第 19 卷 第 9 期:933 ~ 938

doi:10.3772/j.issn.1002-0470.2009.09.010

基于锁的 Cache 一致性协议的硬件优化策略①

蔡 晔②*** 史 岗*

(*中国科学院计算技术研究所 北京 100080)

(**中国科学院研究生院 北京 100080)

摘 要 对一种基于锁的 Cache 一致性协议的开销进行了评估和分析,并结合一款处理器接口的设计,实现了一种分布式内存结构上的远程内存共享机制。该机制能提高处理器机间的通讯性能,降低一致性协议中的通讯延迟,同时通过硬件锁对协议中的同步开销进行优化,避免锁管理器节点陷入处理程序而减少同步等待时间。实际测试结果表明,通过硬件优化的软件 Cache 一致性协议基本操作的性能得到极大的提高,并在实际应用上具有更好的加速比和可扩展性。

关键词 一致性协议,锁,同步,消息传递

0 引 言

对通信性能的依赖以及在处理一致性协议时存在的较大开销,局限了软件 Cache 一致性协议性能的提高及应用。软件 Cache 一致性协议的研究最早针对在消息传递系统上提供便利的基于共享变量的编程模式,从目前的研究现状来看,各种针对协议优化、一致性模型的研究工作已经相对成熟。但是,近年来出现了一些高性能系统域网络(system area network, SAN),如 Memory channel, Infiniband, Quadrics, Myrinet 以及 PCI-express 域网络等,在这些网络上通过硬件支持或相关硬件通讯原语能提供更低延迟的远程内存和更大的传输带宽,因而相关研究工作转向利用这些硬件支持对软件一致性协议进行优化,主要工作有在 Memory channel 上的[1],Infiniband 上的[2,3],PCI-express 上的[4],Myrinet 上的[5]等。即使是在紧耦合系统上,软件一致性协议也有应用的趋势,如采用单元结构的 IBM BLUE-Gene/L 系统中,单个计算单元内集成了 2 个处理器,处理器间通过软件进行一致性协议维护,降低了处理器设计的复杂度和成本。

本文结合一款精简指令集计算机(RISC)处理器接口的设计,实现了一分布式内存结构上的快速远程内存访问机制,设计了相关的硬件对一种基于锁的一致性协议进行了优化。采用这种方法搭建了一个紧耦合的 4 处理机系统,测试结果表明,采用硬件优化的软件 Cache 一致性协议具有更好的加速比和可扩展性。

1 基于锁的 Cache 一致性协议

基于锁的 Cache 一致性协议采用了域存储一致性(scope consistency, ScC)模型。域一致性模型是介于单项一致性(entry consistency, EC)模型和懒惰释放一致性(lazy release consustency, LRC)模型之间的存储一致性模型。它比 LRC 模型更加“懒惰”。在 LRC 模型中,当处理机 P 从处理机 Q 获得锁时,处理机 Q 所看到的(visible)修改操作都被传给处理机 P。但在 ScC 模型中,只有用锁保护起来的区域中所做的修改才会传送给 P。ScC 模型对事件顺序规定如下:

· 在处理机 P 执行获得锁的 Acquire 操作之前,所有相对于锁已执行的访存操作必须相对于处理机 P 执行完。

· 在处理机 P 执行访存操作之前,所有在此之前的 Acquire 操作都已经完成。

一个访存操作相对于一个锁已执行完当且仅当该锁已被释放。

根据存储一致性模型的定义:一个存储一致性模型 M 是一个二元组 $< C_M, SYN_M >$,其中 C_M 是对访存操作的一个分类(或对同步操作的一种描述),

① 863 计划(2006AA010201)和国家自然科学基金(60736012)资助项目。

② 男,1974 年生,博士生,副教授;研究方向:处理器系统设计;联系人,E-mail: caiye@ict.ac.cn

(收稿日期:2008-10-09)

SYN_M 是确定处理机间访存操作执行次序的一种进程间的同步机制，ScC 模型可以如下描述：

1) $C_{ScC} = \{r(x), w(x), acq(1), rel(1)\}$

2) $SO_{ScC}(E(PRG)) = \{(u_i, v_j) \mid acq_i(1) \xrightarrow{PO} u_i \xrightarrow{PO} rel_i(1) \xrightarrow{E} acq_j(1) \xrightarrow{PO} v_j\}^+$ （$\{\cdots\}^+$是$\{\cdots\}$的传递闭包）

在 ScC 模型中，处理机 P_i 发出的访存操作 u 都要通过 2)的执行序列才能被处理机 P_j 所接受，其中，u_i 和 v_j 是冲突访问。如果 u_i 被“$rel_i(1) \xrightarrow{E} acq_j(1)$”定序在 v_j 之前执行，则 u_i 必须在锁 1 所保护的临界区内。

基于锁的 Cache 一致性协议在设计时支持域存储一致性模型和写无效的传播策略，并采用多写协议来避免假共享。该协议最早在文献[6]中提出并在机群系统上进行了实现(JIAJIA)。

2 开销分析及优化策略

2.1 开销分析

基于锁的 Cache 一致性协议的开销主要是在同步点进行一致性维护时以及当访问不命中时从远程节点取数据的开销。基于一致性协议的程序运行时间一般由以下几部分组成：

$$T_{total} = T_{busy} + T_{data} + T_{segv} + T_{syn} + T_{server} + T_{other}$$

其中 T_{busy}为处理器用于计算的有效时间；T_{data}，T_{segv}为处理器数据不命中时的开销，T_{segv} 为进入和退出相应服务程序的时间，一般取决于处理器性能，T_{data} 为在服务程序中远程取数据的时间，一般取决于通信网络性能；T_{syn} 为维护一致性的同步开销，包括获取锁、释放锁、同步等待以及计算和发送 Diff 数据等的开销；T_{server} 为在 Client-server 消息实现机制下处理器处理应答各种消息所需要的时间，这部分开销不包括与处理器空闲时间重叠的部分。而 Tother 为其它未考虑到的微小开销。

为了合理制定优化策略，在采用单个龙芯 2E 处理器作为计算节点的机群上对相关开销进行了量化分析，测试平台为采用 4 台龙芯 2E PC 组成百兆网络机群。每个节点的龙芯 2E 运行在 667MHz，内存为 256MB。并行计算平台采用了机群平台上基于锁的一致性协议的 JIAJIA，在运行库中增加了相应计时部分对开销进行测量和汇总。采用的测试程序为来自 SPLASH 的水分子模拟程序(Water)、海洋模拟程序(Ocean)和 LU 分解程序，来自 NAS 的整数排序问题(IS)，来自 TreadMarks 的旅行商问题程序(TSP)和逐次超松弛法(SOR)。表 1 列出了在一实际 4 机组成的机群上的开销测试结果。

表 1 应用程序开销测试

Apps	T_{total}	$T_{data} + T_{segv}$	T_{syn}	T_{server}
IS 2^{24}	3.776	0.05(1.3%)	0.93(24.6%)	0.04(1%)
LU 2048 × 2048	37.39	3.01(8%)	6.61(17.67%)	2.30(6.1%)
SOR 2048 × 2048,1000 iter	38.41	0.65(1.7%)	8.31(21.6%)	0.76(1.97%)
TSP 19cites	29.8	2.28(7.65%)	3.10(10.4%)	2.15(7.2%)
Water 1728 mol,5 iter	44.43	1.21(2.7%)	7.88(17.7%)	0.46(1%)
Ocean	63.41	31.92(50.3%)	9.76(15.39%)	16.82(26.5%)

由表 1 可知，Ocean 程序由于其共享粒度的特点，所有开销占到总计算时间的 92.3%，其中数据不命中造成的开销占到 50.3%，实际测试 2 机的加速比为 1.01，而四机加速比为 0.29，因而被认为是不适合用软件 Cache 一致性算法计算的程序。从其它几个程序的测试结果来看，同步开销占总时间从最低的 10.4% 到最高的 24.6%，平均为 16.43%。而数据不命中造成的开销平均为 4.27%，T_{server} 开销平均为 3.34%。

针对 T_{data} 以及 T_{syn} 中存在的通讯开销，最基本的优化策略就是提高计算节点间的通信性能，低延迟、高带宽的互连网络能有效地减少 T_{data} 时间，T_{data} 的主要开销为数据不命中时的数据传输时间，以及 T_{syn} 中传送一致性信息的开销，T_{server} 中申请和应答消息的开销等。类似 Ocean 这种程序，在低速网络上加速比很低，原因在于通讯时间占用比例过大。因而现有的很多商用网络如 Infiniband，Myrinet 等都致力于通信能力的提高。本文基于一款 RISC 处理器接口的设计，实现了一种分布式内存结构上处理器间快速的共享内存访问机制，可直接利用处理器组建多处理机系统，通过提高处理器间通信性能对基于锁的软件一致性协议进行优化。

考虑占用比例较大的同步开销,在同步点操作的主要开销在于向锁管理器申请并等待获得锁的开销,由于采用 client-server 机制的消息实现,作为 lock manager 的节点陷入处理程序,记录申请,在适当时候通知申请节点获取锁,并传送相关一致性信息(临界区的 write notice)给申请节点。在这种机制下,申请节点一直处于等待状态,同时锁管理器节点也被迫中止计算,陷入相应的服务处理,严重影响了性能。针对支持远程读写操作的系统,本文实现了一集中式硬件锁管理器,将锁管理器由硬件实现,该管理器支持锁传递时的一致性信息传递,从而可以免去系统中作为锁管理器节点陷入中断服务处理的对应开销,减少 T_{server} 时间,硬件锁管理器的高效率,减少了申请锁处理器的等待时间,从而减少部分 T_{syn} 时间。

2.2　高效分布式内存共享访问机制及系统设计

本文在分布式内存结构上设计了一种耦合层次在处理器片上内存总线上的快速远程共享内存访问机制,并在一款片上集成内存控制器的 RISC 处理器设计时进行了实现。如图 1 所示,映射电路位于处理器 Cache 和内存控制器之间,实现共享空间的地址映射和访问。映射电路将系统内存分为两部分,一部分为本地内存,另一部分作为本地共享内存,被映射到一全局地址空间提供给所有处理器访问。由映射电路区分本地访存操作和远程访存操作,分别发送到本地内存控制器或通过互连网络发送到远程处理器的内存控制器上访问远程内存。

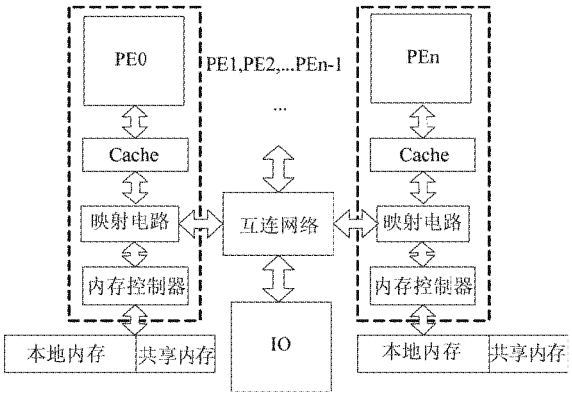

图 1　共享内存访问结构示意图

互连网络可以是点对点网络,用来连接各个处理器和系统 IO 桥。当扩展规模不大时,也可以采用共享总线,因为频繁发生的本地访存操作通过处理器片上专有通道访存,减少了集中共享内存带来的总线冲突,所以采用共享总线也具有较好的可扩展性。为了降低硬件复杂性,互连接口协议设计时不支持 Cache 一致性,就形成了一个典型的不支持 Cache 一致性的访存结构。该映射电路在龙芯 2E 处理器的接口设计时进行了实现,并在接口具体设计时采用了 spilt transaction 技术以及 stream buffer 技术分别对传输协议和内存性能进行了进一步的优化。

根据以上分析,设计了基于龙芯 2E 处理器的多处理机系统,其结构如图 2 所示。多个处理器通过前端总线直接进行互连。通过 IO 桥提供多个处理器对 IO 资源的访问和一个扩展通讯接口 PCI64,同时在 IO 桥上集成了同步控制单元、多处理器中断支持等。

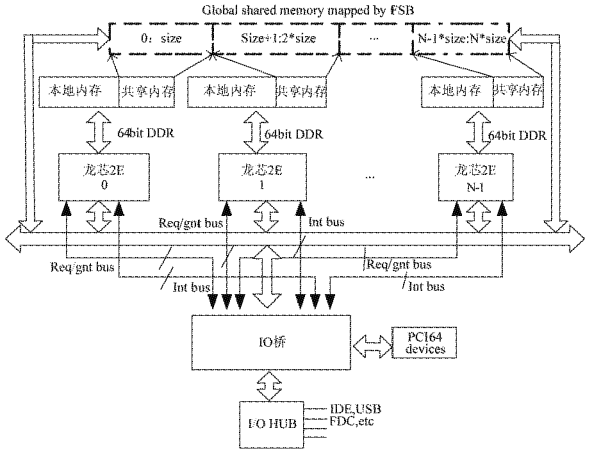

图 2　龙芯 2E 多处理机系统结构框图

系统特点在于多个处理器及 IO 桥通过互连网络连接在一起,并将各自地址空间内一部分((N - 1) * size,N * size)通过映射电路映射在一统一编址的全局共享地址空间内(0,N * size),支持处理器通过 load/store 指令快速访问该共享空间。在这种结构中,处理器可以 cache 访问本地和共享远程内存,但对远程内存访问不提供 cache 一致性支持。在这种结构上,可考虑通过直接远程访存的特性,设计高效的通讯机制来支持软件一致性协议,紧耦合的特点能有效地减少软件一致性协议中的通讯开销。

2.3　集中式硬件锁管理器

本文实现了一集中式硬件锁管理器,可以消除一致性协议在 client-server 实现方式下锁管理器节点陷入处理程序的开销以及减少同步期间的处理器间的等待时间。该硬件锁管理器有以下三个特点:(1)针对在共享存储系统中,同步操作一般通过读-修改-写(read-modify-write,RMW)原子指令和共享变

量实现,如基于 Test—and—Test&Set 原语的锁,基于 LL—SC 指令的锁实现以及分布式的基于链表的队列锁算法(MCS 锁算法)等。本文实现的硬件锁算法采用普通远程读写指令实现,不用借助处理器端的读-修改-写原子指令,因而适用于只支持远程读写操作的系统域网络或类似分布式结构系统中。(2)支持软件 Cache 一致性协议中锁释放时 write notice 的传递。(3)在锁管理器中通过直接存储器存取(DMA)机制来通知远程处理器获得锁,消除了常规锁算法停等阶段查询操作所产生的网络流量。

图 3 所示为硬件锁管理器实现的结构示意图,主要由一个 FIFO 队列以及一个关联的存储空间组成。将常规锁算法中使用的共享计数变量映射成一共享的硬件 FIFO 队列,每个处理器通过环境设置或硬件机制获取一固定标签。处理器申请锁时在本地共享内存分配一同步变量,通过将自己的固定标签和同步变量地址写入 FIFO 来申请锁,FIFO 队列中按顺序存放多个处理器申请。FIFO 的读指针指向当前获取锁的处理器标签及地址,通过硬件 DMA 操作设置对应远程同步变量为处理器标签来通知处理器获得锁。处理器在本地轮询该变量判断是否获得锁。释放锁时通过一个对 FIFO 的读操作改变读指针指向下一个处理器的申请。关联的 Sram 存储空间可以对软件一致性协议进行支持,用于在同步点时对 wrtie notice 进行传递。

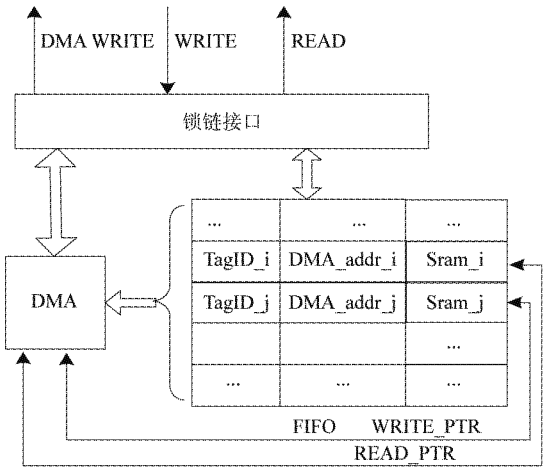

图 3 硬件锁实现框图

与常用锁算法实现类似,支持一致性协议的硬件锁算法的软件工作流程由三个阶段组成。初始化(initialize),申请阶段(acquire)和释放阶段(release)。

1) 初始化(initialize):获取固定标签,在本地共享内存内分配并初始化同步变量。

2)申请锁(acquire):将固定标签和同步变量地址写入 FIFO 队列。

轮询(spin)本地同步变量,等待管理器 DMA 通知获得锁。

从锁管理器 Sram 中读取 write notice 并进行 invalidation 操作

3)释放锁(release):计算 write notice 并合并到管理器 Sram 中读 FIFO 队列,清除自己的申请

由于利用硬件支持将处理器在停等阶段对共享同步变量的查询转换为对位于各自本地内存的同步变量的分布式查询操作,消除了停等阶段的网络流量,硬件 FIFO 同时保证了获取锁的公平性。每个处理器获取锁仅需 3 次网络操作(一次申请的写操作,一次 DMA 通知的写操作,一次释放锁的读操作)。通过多个 FIFO 队列可以实现对多个锁的支持,每个锁占用一个系统共享地址。

对比基于 Test—and—Test&Set 原语的锁实现,一次锁操作最坏情况下产生 $O(p^2)$ 量级的网络交通量,而基于 LL—SC 指令的锁实现,一次锁操作最坏情况下产生的网络交通量为 $0(p)$ 量级。以及采用类似思路的分布式 MCS 算法,一种基于链表的队列锁,支持在本地存储器上忙等待,在没有竞争的情况下,算法需要一次 Fetch&Store 来获得锁,一次 Compare&Swap 来释放锁。在竞争激烈的情况下,长度为 p 的忙区间内一共会产生$(3p-1)$次网络传输,其中 RMW 操作为 $(p+1)$ 次,在最坏的情况下则为 $(4p-2)$ 次网络传输,其中 RMW 操作为$(2p)$次。本文实现的硬件锁机制则消除了停等阶段的网络流量,长度为 p 的忙区间内网络流量固定为 $3p$ 次,与 MCS 算法比,采用常规读写操作完成,软件实现简单,不需要维护复杂的数据结构,并能通过关联的存储空间实现对一致性协议的支持,从而有效地减少 Cache 一致性协议在实现时较大的同步开销。

3 性能分析

在设计的多处理机系统中,在系统 IO 桥上实现了硬件锁,IO 桥采用了 Altera 的 Stratix II FPGA 实现。在该硬件平台上对本文提出的优化策略进行评估。利用多处理器间快速共享内存通讯机制实现了一虚拟网卡驱动(Vnet on TCP/IP),并利用其实现了一 JIA-NUMA-LIB 库,利用硬件锁机制对软件一致性协议的实现进行了优化。同时基于 Vnet 移植了 JIAJIA 和 MPI 运行库(mpich2-1.0.5p4)用于性能对比。分别对 JIA _ NUMA _ LIB 的基本函数性能以及

实际应用程序的性能进行了测试和分析。通过100M 网络机群与 Vnet 的性能测试对比,可获得在通讯协议相同的情况下,由硬件通讯性能提升带来优化效率;通过对比采用硬件锁机制的 JIA _ NUMA _ LIB 库与不采用该机制的 JIAJIA(Vnet)库的性能对比,可获得采用硬件锁对协议进行优化带来的性能提升。

测试操作系统采用了 linux2.6.18,在测试中,所有库和应用程序编译选项为 gcc-O2。系统中每个处理器的工作频率为 667MHz,内存为 256MB。

3.1　基本操作性能

JIA _ NUMA _ LIB 库在实现时将一致性协议支持封装在同步函数中,主要的同步函数有 JIA _ NUMA _ LOCK、JIA _ NUMA _ UNLOCK 和 JIA _ NUMA _ BARRIER 等函数,用户在程序设计时可直接调用而不用关心一致性协议实现的细节。如代码 1,2 中的程序片段所示,利用两个处理器对这些函数的基本操作性能进行了测试。代码 1 中,在一个处理器内存上分配了一共享变量,然后两个处理器依次利用锁操作对共享变量进行操作,通过计算在临界区时间可以评估处理器对远程共享变量操作所需要的一致性维护的相关开销。代码 2 则评估了当一个处理器等待另一个处理器更新共享变量,所需要的栅栏同步基本操作的性能。代码中 jia _ numa _ wait 函数为不支持一致性的栅栏操作。

```
float startt, endt;
int i;
double * ptr;
jia _ numa _ init(argc,argv);
ptr = (double * )jia _ numa _ alloc(sizeof(double));
if (jiapid = = 0) * ptr = 0.0;
jia _ numa _ wait();
startt = jia _ clock();
for (i = 0;i < 100000;i + + ){
jia _ numa _ lock(1);
    * ptr = * ptr + 1;
    jia _ numa _ unlock(1);
    }
jia _ numa _ wait();
endt = jia _ clock();
```

代码 1　锁基本操作性能测试代码

```
float startt, endt;
int i;
double * ptr,data;
jia _ numa _ init(argc, argv);
ptr = (double * )jia _ alloc(sizeof(double));
jia _ numa _ wait();
startt = jia _ clock();
for(i = 0; i < = 100000; i + + ){
    if(jia _ pid = = 0)
        * ptr = i;
    jia _ numa _ barrier();
    data = * ptr;
    jia _ numa _ wait();
}
endt = jia _ clock();
```

代码 2　栅栏基本操作性能测试

图 4 给出了相关测试程序在 100MB 机群,采用 Vnet 网络驱动 JIAJIA 库以及 JIA _ NUMA _ LIB 库的测试结果。由图可知,支持一致性协议的锁操作和栅栏同步操作由于通讯性能的优化分别带来了 35.5 倍和 6.7 倍的性能提升。采用硬件锁的优化效率也分别提高了 52.9%和 21.5%。

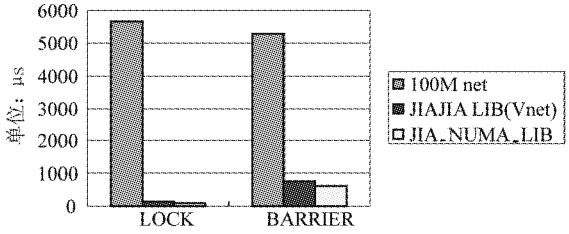

图 4　基本操作性能对比

3.2　应用程序性能评估

对节 3.1 中使用的 5 个实际应用程序进行了性能测试。分别在采用龙芯 2E 处理器的 100M 机群系统上的 JIAJIA 库,在本文设计的硬件平台上采用基于 Vnet 驱动的 JIAJIA 库,以及在本文设计的硬件平台上采用 JIA _ NUMA _ LIB 库进行测试。测试的加速比结果见图 5(a),测试中采用了 4 个处理器。

从测试结果可知,由于通讯能力的提高,在四处理器时,原来适用于软件一致性的测试程序以及Ocean程序都获得了性能提高,而基于硬件锁的优化机制在所有应用程序的测试对比中也获得了性能提高。测试中,性能提高最显著的是Ocean程序,从在机群上的四机加速比由于通讯优化从 0.288 提高到 1.734,基于硬件锁的优化策略又进一步将性能提高了 22%,加速比达到 2.1,因而通过硬件优化的软件 Cache 一致性协议扩大了应用的范围,原来不适用的程序获得了可接受的加速比。另针对 Ocean 程序在

相同硬件条件下与在 MPI 环境的测试结果表明(图 5(b)),与精雕细刻的基于消息传递的编程机制相比,对 Ocean 程序而言,JIA _ NUMA _ LIB 在性能上仍有 22.3%左右的差距,但已经获得较接近的性能。

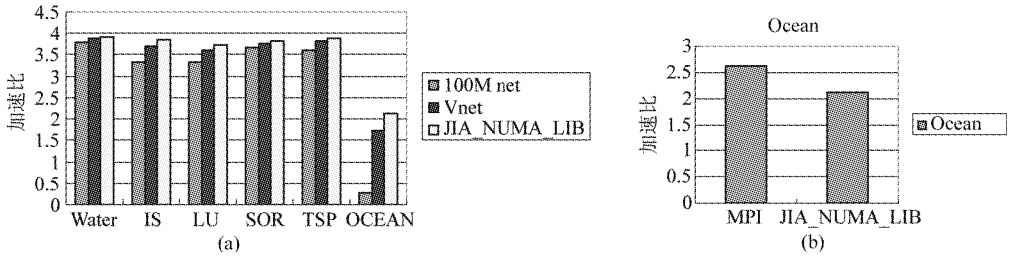

图 5 应用程序性能对比

4 结 论

本文对基于锁的 Cache 一致性协议的开销进行了分析,在这基础上提出了相关的硬件优化策略。通过一款 RISC 处理器接口的设计,基于分布式内存实现了一种高效的共享内存访问机制,并实际构建了一紧耦合的多处理机系统,通过通讯能力的提高对一致性协议性能进行优化,同时利用系统支持远程读写的特性实现了一硬件锁机制,通过该机制对一致性协议的实现进行了优化,避免了锁管理器节点陷入处理程序的开销以及减少同步期间的等待时间。最后对软件 Cache 一致性协议实现的基本操作以及相关应用程序的性能进行了评估。测试结果表明,通讯性能的提高对一致性协议的性能优化效果明显,在硬件成本日趋降低的趋势下,是一种有效的优化方法,另外通过硬件锁机制对一致性的实现进行优化的方法也取得了较好的效果。本文的工作表明,通过硬件优化的软件 Cache 一致性协议具有更好的加速比和可扩展性,同时改变了协议应用的局限性,扩大了应用的范围。

参考文献

[1] Kontothanassis L, Stets R, Hunt G, et al. Shared memory computing on clusters with symmetric multiprocessors and system area networks. *ACM Transactions on Computer Systems*, 2005,23(3):301-335

[2] Noronha R, Panda D K. Reducing diff overhead in software DSM systems using RDMA operations in infiniband. In: Proceedings of the 2004 Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations, and Technologies, San Diego, USA, 2004

[3] Noronha R, Panda D K. Designing high performance DSM systems using infiniband features. In: Proceedings of the 2004 DSM Workshop Conjunction with 4th IEEE/ACM International Symposium on Cluster Computing and the Grid, Chicago , USA, 2004. 467-474

[4] Noronha R, Panda D K. Can high performance DSM systems designed with infiniband features benefit from PCI-express? In: Proceedings of the 2005 DSM Workshop Conjunction with the 5th IEEE/ACM International Symposium on Cluster Computing and the Grid, Cardiff, UK, 2005. 945-952

[5] Buntinas D, Panda D K, Gropp W. NIC-based atomic operations on myrinet/GM. In: Proceedings of the 2002 SAN-1 Workshop Conjuction with High Performance Computer Architecture, Boston, USA, 2002

[6] Hu W W, Shi W S, Tang Z M. JIAJIA: a software DSM system based on a new cache coherence protocol. In: Proceedings of the 1999 International Conference on High Performance Computing and Networking Europe, Amsterdam, The Netherlands, 1999. 463-472

Optimization of lock based Cache-coherence protocol by hardware

Cai Ye[*,**], Shi Gang[*]

([*] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080)

([**] Graduate University of Chinese Academy of Sciences, Beijing 100080)

Abstract

The overhead of a lock based cache coherence protocol was evaluated and analyzed. To reduce the major communication overhead, a fast shared memory-accessing method was proposed on a distributed memory architecture and implemented in design of a reduced instruction set computer (RISC) processor. And a hardware lock scheme was also proposed to reduce the synchronization overhead. Based on this scheme the overhead of trapping to service program of lock manager machine can be eliminated and the waiting time during the synchronization period can also be reduced. The experimental result shows that the lock-based cache coherence protocol which optimized by the proposed hardware schemes has the better parallel speedup and scalability.

Key words: coherence, lock, synchronization, message passing