CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 44 卷第 11 期 2022 年 11 月
Vol. 44, No. 11, Nov. 2022

文章编号:1007-130X(2022)11-1901-08

面向众核系统的层次化栅栏同步机制*

臧照虎,李　晨,王耀华,陈小文,郭　阳
(国防科技大学计算机学院,湖南 长沙 410073)

摘　要:同步操作在保证多核处理器线程的数据一致性和正确性等方面起着重要作用。随着处理器内核数量的不断增加,同步操作的开销也越来越大。栅栏同步是并行应用中多核同步的重要方法之一。软件同步方法通常需要数千个周期才能完成多个内核之间的同步,这种高延迟和串行化同步会导致多核程序性能的显著下降。相比于软件栅栏同步方法,硬件栅栏能够实现较低的同步延迟,然而传统集中式硬件栅栏的可扩展性有限,难以适应众核处理器系统的同步需求。面向众核处理器提出了一种层次化硬件栅栏机制——HSync,它由本地栅栏单元和全局栅栏单元组成,二者协调配合,以实现低硬件开销的快速同步。实验结果表明,与传统的集中式硬件栅栏相比,层次化硬件栅栏机制将众核处理器系统性能提高了 1.13 倍,同时网络流量减少了 74%。

关键词:硬件同步;栅栏;众核系统;并行计算
中图分类号:TP393　　**文献标志码**:A
doi:10.3969/j.issn.1007-130X.2022.11.001

A hierarchical hardware barrier synchronization design for many-core processors

ZANG Zhao-hu, LI Chen, WANG Yao-hua, CHEN Xiao-wen, GUO Yang
(College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China)

Abstract: Synchronization plays an important role in ensuring data consistency and correctness of multicore processor threads. As the number of processor cores increases, the cost of synchronization increases. Barrier synchro-nization is one of the effective methods for multi-core synchronization in parallel applications. Software synchronization methods typically require thousands of cycles to complete synchronization among multiple cores. This high latency and serialization synchronization can result in significant performance degradation of multicore programs. Compared with the software barrier synchronization method, the hardware barrier can achieve lower synchronization delay, but the scalability of the centralized hardware barrier is limited and it is difficult to adapt to the multicore processor systems. This paper proposes a hierarchical hardware barrier mechanism called HSync for multicore processors. It consists of local and global barrier units, which work together to achieve fast synchronization with low hardware overhead. The experimental results show that the hierarchical hardware barrier mechanism improves the performance of the multicore proces-sor system by 1.13 times and reduces network traffic by 74% compared with the traditional centralized hardware barrier.

Key words: hardware synchronization; barrier; many-core processors; parallel computing

* **收稿日期**:2021-11-22;**修回日期**:2022-03-22
基金项目:国防科技大学科研计划(ZK20-04)
通信作者:李晨(lichen@nudt.edu.cn)
通信地址:410073 湖南省长沙市国防科技大学计算机学院
Address: College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, Hunan, P. R. China

1 引言

当前,在多核及众核处理器中,大量并行应用程序和程序模型依赖于线程之间的同步来确保一致性和正确性[1,2]。因此,这类应用程序的性能受制于同步原语的延迟,随着并行处理器或计算数据规模的增大,这种依赖性变得越来越大。栅栏同步是一个重要的同步原语,它会阻塞线程,直到所有线程都到达同步点,继而恢复执行。栅栏通常用于将并行应用分离成不同的执行阶段,以确保在对应阶段所有线程完成之前,下一阶段不会启动。

为了实现栅栏同步,研究人员提出了许多基于软件的方法[3,4]。通常,所有线程共享一个变量。当每个线程到达同步点时,访问内存并修改该变量,使其值加 1,然后进入等待状态。当变量的值达到预设的同步线程数时,所有线程都将恢复执行。软件栅栏同步方法仍然需要基于同步原语,如信号量和锁[5,6],利用这些同步原语,通过共享内存实现栅栏同步。由于共享变量存储在内存中,每次修改共享变量都需要访问内存,因此长内存访问延迟可能成为系统的瓶颈。

利用共享内存实现的栅栏的开销主要来自访问共享内存的原子操作,因此可以利用 Cache 一致性协议来实现[7,8]。然而,一些多核处理器可能不支持硬件 Cache 一致性协议,因此这些系统无法采用这种方法[9]。为了降低这种软件同步开销,研究人员提出了几种基于硬件实现的栅栏同步方法[10-14],典型的是在处理器上集成一个集中式硬件栅栏单元,用于处理来自所有内核的栅栏同步请求。与软件栅栏同步相比,采用硬件设计的集中式栅栏同步方法效率更高。然而,当它运行在多核甚至众核处理器系统上时,这种集中式栅栏同步方法可能会导致网络拥塞[15]。陈小文[16]提出一种协同通信方法,用于在基于 mesh 的众核处理器中实现高效和高扩展性的 all-to-all 栅栏同步机制,该方法使用片上网络传递同步报文,但仍会占用片上网络资源,可能在网络中形成热点。

因此,本文提出一种称为 HSync(Hierarchical Synchronization)的层次化硬件栅栏同步设计。HSync 建立在分层网络上,可以减少栅栏同步原语的同步流量,并且易于扩展以满足众核处理器的需求。实验结果表明,与传统的基于集中式硬件栅栏的栅栏同步相比,HSync 实现了 13%的性能提升和高达 74% 的同步报文减少。

2 背景

研究人员在过去的几十年中已经提出了许多方法,例如在软件中利用原子操作实现同步[17]。软件同步方法虽然因其实现简单而被广泛使用,但是,构建于硬件共享内存系统之上的软件实现的栅栏同步操作延迟很高,内核间通信的流量和延迟都随着同步参与者数量的增加而呈指数增加[18],这导致了同步数百个线程的效率极低。

当前已有研究人员使用专用硬件来提高栅栏同步性能,但这些方法也存在一些缺点,例如缺乏可扩展性或对并发栅栏的支持有限。从本质上来说,栅栏同步原语是一个与门,在硬件实现中,栅栏的实际性能受到芯片中连线延迟的限制。Hsu 等[19]提出了一个多级随机交换网络,通过组合交换机中的数据包来有效处理软件栅栏的同步流量,以缓解网络中的热点拥塞。Monchiero 等[20]提出使用硬件模块来减少 CMP(Chip MultiProcessor)系统中核心的等待时间,该模块集成在内存控制器中,称为 SOB(Synchronization Operation Buffer)。SOB 管理本地核心对共享变量的轮询,避免产生网络流量及访存。众核系统中各核心也可以通过片上网络进行同步信息的交换。Giannoula 等[21]提出的 SynCron(Synchronization)适用于 NDP(Near-Data-Processing) 架构,此架构将处理核心置于数据附近,而不是将数据搬移至处理器中。该方法的优势在于将信号量、锁、栅栏及条件变量 4 种同步原语集中到一个硬件中,但是也因此带来了更大的开销,降低了灵活性,而且没有应用到实际当中。本文方法依托于国防科技大学的众核处理器项目,已应用于实际芯片,在减少网络流量,降低时延,降低开销的同时,拥有较强的扩展性。Hetland 等[22]基于 Intel 的 HARP(Hardware Accelerator Research Program)多芯片架构平台提出了 Harp Barrier 的同步方法,此方法的优势在于利用这种混合架构的 FPGA 及缓存一致性接口实现栅栏,但是其缺点在于难以适用于一般的众核处理器。

综上,软件栅栏会带来较高的时间开销,而集中式硬件栅栏同步会占用片上网络,可能造成网络拥塞及热点问题。因此,为了避免上述问题,本文设计了基于层次化网络进行消息传递的方法。层次化网络最大限度地减少了跨结点的通信报文,并在高并发场景下具有高性能。为了保证设计的正

确性,参考验证方法学[23,24]对 HSync 进行了模块级完善和系统级验证。

本文是基于国防科技大学自主研发的高性能通用 DSP(Digital Signal Processor)提出的 HSync。该款 DSP 具有 24 颗 DSP 核心,所有的核心被分为 4 个结点,6 个核组合成 1 个结点。HSync 在每个结点内设置一个本地栅栏单元 LBU(Local Barrier Unit),本结点的所有核都与之连接,由其负责本结点的栅栏同步请求。全局栅栏单元 GBU(Global Barrier Unit)设置在结点之间,所有本地栅栏单元都与其相连,在完成本地栅栏同步后,再进行全局栅栏同步。

HSync 使用独立的栅栏网络,网络流量主要有 2 种类型,即本地栅栏同步流量和全局栅栏同步流量。如果没有分层设计,即使是本地栅栏同步流量也需要跨全局栅栏网络传播,这大大增加了全局栅栏网络的压力,可能导致网络拥塞和热点问题。同时,随着信息传输跳数的增加,本地栅栏同步的延迟会进一步增加。因此,分层栅栏结构可以显著减少全局栅栏同步流量,降低本地栅栏同步延迟。

3 HSync 设计

3.1 总体结构

HSync 是多核处理器中栅栏同步的有效解决方案,可以提高并行程序性能,降低网络开销,简化编程难度。该方案的设计主要是基于一定数量的计算核心组成一个基本单元,称为结点,若干结点构成一个处理器。HSync 的结构具有很强的可扩展性,便于应用在各种多核或众核微处理器上。程序员在使用该装置时只需要确定每个结点需要同步的核数及所有结点同步的总核数,并且使用同一个同步 ID 即可。

HSync 使用了高度协作的两级栅栏同步单元。如图 1 所示,栅栏同步单元包括结点内部的本地栅栏单元和结点之间的全局栅栏单元,本地栅栏单元连接全局栅栏单元。

在 HSync 栅栏同步操作中,单个核只能发起一个栅栏同步操作,但可以同时参与多个不同的栅栏同步。对于栅栏同步请求,每个结点内部的线程首先进行同步,然后全局栅栏单元收集所有结点的同步信息进行处理。这种分层处理方式汇集了同步信息,减少了网络中的数据包数量,可以更快地完成结点内的栅栏同步。

本地栅栏单元(LBU)由寄存器和相应的控制

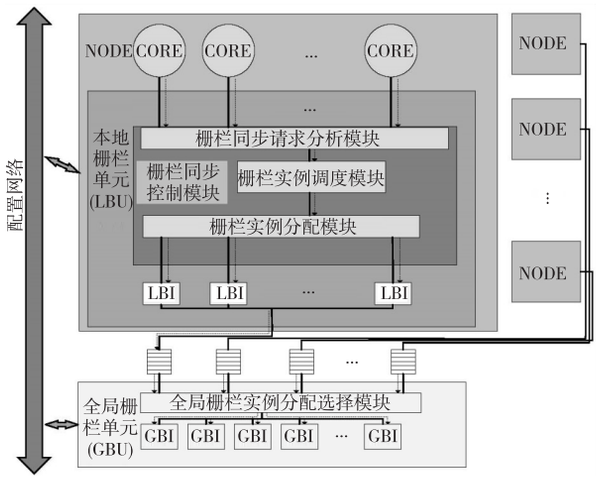

Figure 1 Overall structure of HSync

图 1 HSync 的整体结构

逻辑组成,位于每个结点内,负责处理结点内的栅栏同步请求。本文在 LBU 中设置了若干本地栅栏实例 LBI(Local Barrier Instance),每个硬件栅栏实例用于记录一次栅栏同步的所有信息。LBU 还包含了处理同步请求、分配 LBI 及向内核发送释放信号的逻辑。LBU 的结构如图 1 所示,主要由栅栏同步请求分析模块、栅栏实例调度模块、栅栏实例分配模块和一些 LBI 组成。

为了支持多个线程并发的栅栏请求,本文在每个 LBU 中设置了多个 LBI,并可根据系统规模进行调整。LBI 主要包括以下功能:(1)同步 ID 区分不同的同步请求;(2)记录结点内需要同步的线程总数;(3)记录整个处理器中需要同步的线程总数;(4)使用进程向量寄存器记录已经到达的线程;(5)一些配置功能,例如超时寄存器为同步请求预留了最长的保持时间。

HSync 中为每个栅栏实例提供了一些配置寄存器,用户可以通过设置这些寄存器来控制每个 LBI。比如,通过设置栅栏超时寄存器,可以调整 LBI 的最长占用时间,以适应不同的工作环境。如果确定不需要配置 LBI,可以去掉部分配置寄存器,以减少设计的面积开销。

全局栅栏单元(GBU)用于处理结点间的栅栏同步请求。与 LBU 的主要区别在于其内部有多个 FIFO,用于临时存储来自 LBU 的消息。GBU 包含多个全局栅栏实例 GBI(Global Barrier Instance),GBI 与 LBI 的组成类似。GBU 首先接收来自 LBU 的消息,对其进行解码以跟踪全局栅栏同步,并在所有线程到达时向 LBU 发送释放信号。GBU 的具体结构如图 2 所示。

为了增强本文栅栏设计方案的可配置性,以及

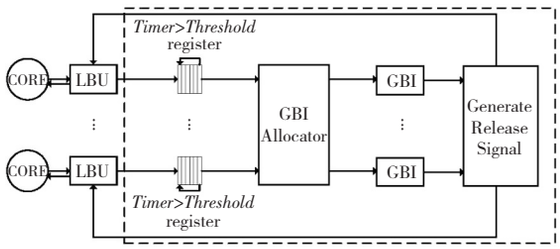

Figure 2 Structure of GBU

图 2 全局栅栏单元的结构

协助程序员在出现问题时定位错误,本文添加了一些用于观察和控制栅栏的寄存器,包括状态寄存器组和配置寄存器组。状态寄存器用来存储栅栏机制的当前状态,程序员可以通过读取这些寄存器的值来检查机制的运行状态;栅栏的配置信息存储在配置寄存器中,程序员将特定值写入这些寄存器以配置栅栏。

全局栅栏映射寄存器:该寄存器存储了全局栅栏实例与栅栏同步请求 ID 的映射关系。每个栅栏同步请求 ID 对应一个全局栅栏实例。这个映射关系是由硬件根据空闲栅栏实例自动建立的,程序员可以通过读该寄存器来获取该栅栏请求所访问的栅栏实例。全局栅栏实例有效寄存器:默认所有位均为 1。当使用栅栏实例时,其对应值设置为 0。当栅栏同步完成后,栅栏实例被释放,寄存器恢复为全 1。全局错误寄存器:当该寄存器设置为 1 时,表示栅栏处理发生异常,例如超时错误,将发送中断信号通知系统;设置为 0 时,则中断关闭。软复位寄存器:对该寄存器的低位写 1 将复位对应结点的栅栏单元。阈值控制寄存器:该寄存器的值表示该结点发送的同步请求存在于缓冲区头部的时间。超时后,头部请求将放置在缓冲区的末尾。本地栅栏单元中的寄存器组与全局栅栏单元中的类似,但没有阈值控制寄存器。

全局栅栏实例和本地栅栏实例也包含状态寄存器和配置寄存器。栅栏状态寄存器:该寄存器存储了栅栏同步请求的相关信息,包括同步请求 ID、参与同步的总核数等。本地栅栏实例还包含参与同步的内核总数。栅栏实例状态向量寄存器:表示当前到达的核心向量。超时寄存器:表示一个栅栏请求可以等待的最长时间,超时后此栅栏实例将被释放。超时计数寄存器:当一个栅栏请求被分配给一个栅栏实例时,此寄存器会开始计数,记录此同步请求的等待时间,当超过超时寄存器的值时,会释放栅栏,以避免长时间等待。栅栏实例错误使能寄存器:将此寄存器设置为 1 后,如果对应 LBI 在处理栅栏请求中发生错误,将通知系统处理错误并做出其他响应。栅栏实例错误寄存器:用于记录发生错误时到达的核心向量,以方便于系统查询和定位错误。

3.2 HSync 工作流程

一般来说,可以将来自核心的同步请求分为 2 类:(1)本地同步:指只有结点内的处理器核上的线程参与的栅栏同步。当一个线程发送栅栏请求时,LBU 接收请求并选择一个空闲的 LBU 来记录此栅栏同步的详细信息。当 LBU 收到来自其他参与同步的内核的请求时,它会更新之前选择的 LBI。直到所有的核都到达,LBI 将被 LBU 复位,并向参与同步的核发送释放信号。如果参与本地同步的核数为 1,则直接将请求发送给 GBU 进行处理。(2)全局同步:指参与同一次同步的线程位于不同结点。不同结点的核首先完成本地栅栏同步,然后将相应的同步信息发送给 GBU。与 LBU 流程类似,GBU 会为此请求分配一个空闲的 GBI 记录同步信息。当所有参与同步的核到达时,GBU 将重置 GBI,并向 LBU 发送释放信号。LBU 将发布信息转发给每个内核,所有线程恢复执行。

栅栏同步请求分为本地和全局,可以减少本地同步报文在网络中的传输时间。同时,层次化的多级设计减少了全局同步的报文数量。一次完整栅栏同步的基本流程如图 3 所示。本文假设所有内核同时参与相同的栅栏同步。

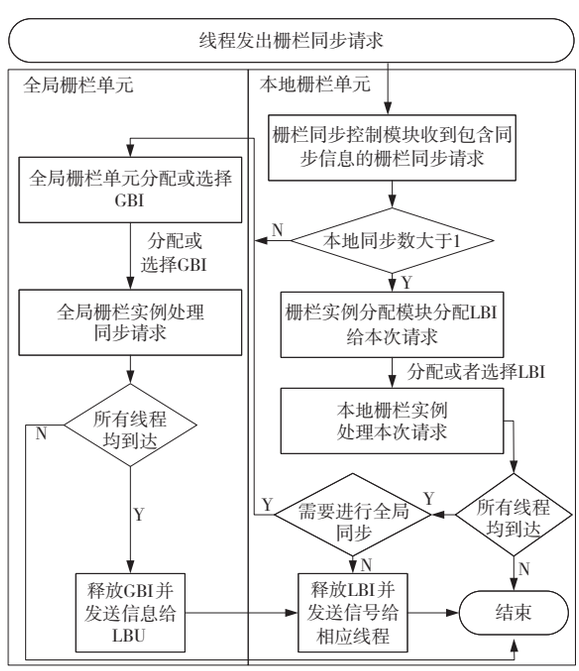

Figure 3 Flow chart of barrier synchronization request

图 3 栅栏同步请求流程

3.3 编程接口

HSync 使用体系结构中已有的 load 指令(访问特定范围的地址)访问栅栏空间的方式发出同步请求,线程阻塞在此指令,直到所有线程都到达栅栏。这种方法的优点是不需要添加新指令或修改指令集。栅栏同步请求的具体信息编码在 load 指令的地址参数中。栅栏空间地址包含 3 个字段:*SyncID*(同步请求 ID)、*LocalNum*(本地线程数)和 *GlobalNum*(全局线程数),它们各自的字段宽度根据实际系统配置确定。

下面给出了栅栏操作以及访问栅栏内部寄存器的软件接口:

int *bar_req* (unsigned int *SyncID*, unsigned int *LocalNum*, unsigned int *GlobalNum*);

int *set_bar_reg*(unsigned int *RegName*, unsigned int *Value*);

int *get_bar_reg*(unsigned int *RegName*);

程序员需要向发起栅栏请求的函数传递 3 个参数:表示栅栏同步请求的 ID 的 *SyncID*、存储本地结点参与同步的线程数 *LocalNum* 和存储参与此同步的全局线程总数 *GlobalNum*。当栅栏同步正确完成时,该函数返回一个大于零的值。整个同步机制对程序员是透明的。当内核发送同步请求时,该机制自动分配空闲的 LBI 和 GBI,无需程序员指定。

为了提高方案设计的灵活性和稳定性,本文还在 HSync 内部设置了一些配置寄存器供程序员访问。设置内部寄存器的系统调用需要提供 2 个参数:*RegName*(目标寄存器的名称)和 *Value*(要设置的值)。获取状态寄存器值的函数只需要提供寄存器地址。

3.4 可扩展性

本节讨论如何对本文提出的方案进行扩展,以适用于更大规模的系统,便于 HSync 应用到不同的场景中。HSync 可以通过 2 种方式应用于更大的系统。第 1 种方式是增加 LBU 中 LBI 的个数和 GBU 中 GBI 的个数。这种方法比较简单容易实现,但是在系统规模巨大时会造成很大的硬件开销,且在实际程序中很少会有并发栅栏请求占满 LBI 或 GBI 的情况。在实际测试中发现确实如此,每个结点通常不会同时有大量不同 ID 的同步请求,因此大多时候有一部分 LBI 或 GBI 并未被使用。第 2 种方式使用固定数量的 LBI 和 GBI,当同步请求数超过 LBI 或 GBI 数量时,在同步装置中增加 FIFO 来缓存同步请求。

为了减少资源浪费,LBI 和 GBI 的数量可设置为比核数更小的值,因此如果 LBI 或 GBI 被全部占用,则来自内核的新同步请求可能会阻塞暂存于同步请求的 FIFO。在这种情况下,会产生死锁。为了解决这个问题,本文修改了传统的 FIFO,增加了一些控制逻辑。当 FIFO 中的数据在一定时间内没有被移除时,后续的请求将首先对 FIFO 头进行处理。这种方法不仅减少了 LBI 和 GBI 的数量,而且对本文设计的性能没有显著影响,大大增强了设计的可扩展性,减少了资源浪费。

4 实验

为了衡量栅栏同步操作对于并行应用程序的影响,本文使用 2 个计算核进行实验。如图 4 所示,随着核数的增加,软件同步方法引起的同步开销迅速增加。当有 32 个内核在运行矩阵乘法运算时,接近 66% 的执行时间是消耗在各进程同步上的,对于需要多核栅栏同步的运算,随着核数增加,栅栏同步操作占比会更大。对于不同运算,栅栏同步操作所占比例不同,但随着核数增多,占比增大的趋势是相同的。

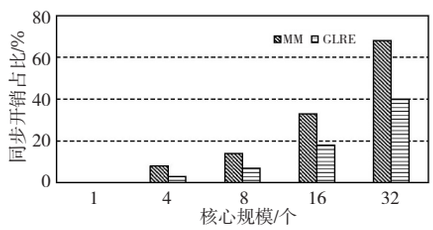

Figure 4 Proportion of time cost for software synchronization operation under different core scales

图 4 不同核心规模下软件同步占总执行时间比例

本文采用 Verilog 硬件描述语言实现了 HSync 硬件栅栏设计,并通过将此设计集成到系统级环境中得到实验结果。本文在 Linux 系统上使用仿真软件平台对 HSync 进行了测试。实验中使用的 24 核系统,一共 4 个结点,每个结点具有 6 个核心。

为了横向比较 HSync 的性能,本文共实现了 3 种栅栏设计:软件栅栏同步方法 SW(SoftWare)、集中式栅栏设计 FlatBar(Flatten Barrier)和本文提出的层次化栅栏 HSync。依托于国防科技大学自主研发的众核 DSP 项目,本文实现了这 3 种栅栏方案并将其用于本文实验。使用硬件实现的栅栏机制相比于基于软件的机制具有较大的优势:本文中软件实现的栅栏利用了共享内存,所有参与同

一次栅栏同步的线程共享一个计数器,其中每个线程在到达栅栏时递增该共享计数器,通过信号量控制该计数器的修改并等待,直到该计数器达到预定值,即所有线程均到达栅栏,所有线程恢复执行。FlatBar是一种无层次的硬件栅栏设计,结点内部的栅栏请求在结点内部进行处理,而位于不同结点的所有线程的栅栏请求将集中到一处进行处理。

为了全面地评估HSync,本文使用了几种不同的测试方案,包括理论性能测试、随机激励测试和实际应用测试。为了测试本文方案的理论性能,首先建立了一个合成流量模型。主要做法是调整本地同步请求在全体同步请求中所占的比例。在此测试中,分别选取了5个典型值:0,25%,50%,75%和100%。基于这个条件,随机产生相应的栅栏同步请求,并不断地发送到同步单元,同时监测网络中的流量并记录同步消耗的时间。本文还选择了几种常用的并行计算核,如FFT、一般线性递推方程等。本文评估中所使用的计算核如表1所示。

Table 1 Computational kernels used in this experiment

表1 本实验所使用的计算核

简称	全称
MM	Matrix Multiply
FFT	Fast Fourier Transform
ICCG	Incomplete Cholesky Conjugate Gradient
IP	Inner Product
FS	First Sum
FD	First Difference
PD	Planckian Distribution
GLRE	General Linear Recurrence Equations

本文使用合成流量模型和一些并行计算核对HSync进行了全面测试。为了清楚地了解栅栏同步所占用的处理器执行时间,在实验平台上测量了1次栅栏同步的时间。图5表明,硬件栅栏同步相比软件实现耗时明显减少了,本文的层次化栅栏设计耗时相比集中式也有明显降低。

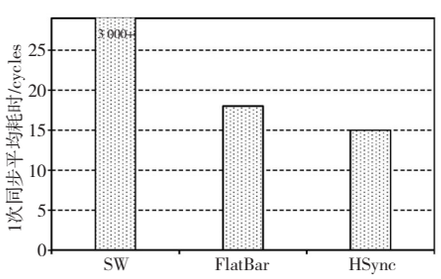

Figure 5 Average time spent in one barrier synchronization

图5 1次栅栏同步的平均耗时

4.1 不同栅栏对程序执行时间的影响

从图6可以看出,随着本地栅栏同步请求比例的增加,由于本地的网络延迟更低,因此性能提升更加明显。图7为以集中式栅栏为基准不同栅栏方案下各并行计算核的性能。横坐标代表使用的计算核,纵坐标代表归一化的执行时间。由于降低了本地栅栏的同步延迟,HSync在实际应用中相对于无层次硬件栅栏方案也有一定的改进。

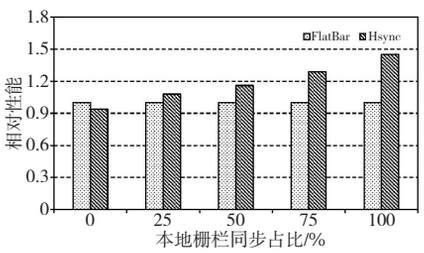

Figure 6 Performance of different barrier schemes in synthetic traffic models

图6 合成流量模型中不同栅栏方案的性能

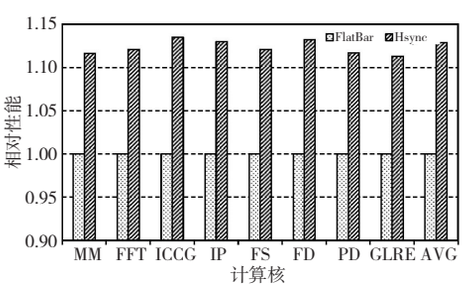

Figure 7 Performance of different barrier schemes in parallel computing kernels

图7 并行计算核中不同栅栏方案的性能

4.2 不同栅栏对网络流量的影响

受益于层次化的设计,结点内的栅栏同步不会在结点之间的网络中产生报文。对于全局同步,栅栏请求会被结点中的本地栅栏单元处理,合成一条报文,然后在结点间的网络中传输。因此,栅栏同步的网络流量得到了很大的减少。图8描述了合成流量模型网络中的归一化报文数量。横坐标表示本地栅栏同步在所有同步请求中的比例。正如预期的那样,随着本地栅栏同步所占比例逐步增加,结点间网络中传输的数据包总数则逐步减少。当所有数据包都是本地栅栏同步请求生成时,数据包数量最少。得益于层次化的结构设计,可以看到图9中实际的并行计算核中网络流量显著减少,因为网络中数据包合并后大大减少了报文数量。

4.3 LBI和GBI数量对设计的影响

在本文的设计中,LBU和GBU分别用于分析并处理本地同步请求和全局同步请求,它们包含的

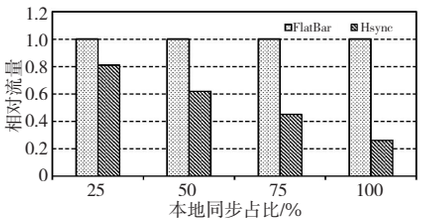

Figure 8 Network traffic of different barrier schemes in synthetic traffic models

图 8 合成流量模型中不同栅栏方案网络流量

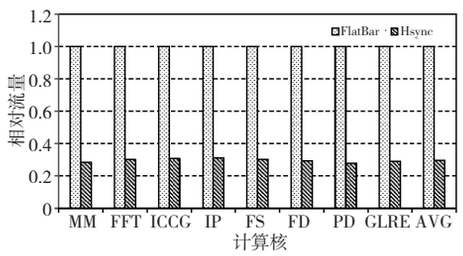

Figure 9 Network traffic of different barrier schemes in different computing kernels

图 9 并行计算核中不同栅栏方案网络流量

LBI 和 GBI 的数量可能会影响 HSync 执行栅栏同步的性能。因此,为了分析不同数量的 LBI 和 GBI 对 HSync 性能的影响,设计了几组实验。实验基本配置由 4 个结点组成,每个结点包含 8 个核。通过在不同的 LBU 和 GBU 配置下执行相同的应用程序,比较实际执行时间得到实验结果。图 10 显示了在每个结点中设置不同 LBU 数量对性能的影响。结果表明,当 LBU 数量为 3 时,HSync 对栅栏同步的加速比达到较高值。图 10 表明,GBU 的数量在 4～6 时达到较高的加速比。根据此实验结果,可以确定 HSync 中 LBI 和 GBI 的最佳数量,以实现性能和硬件开销之间的平衡。

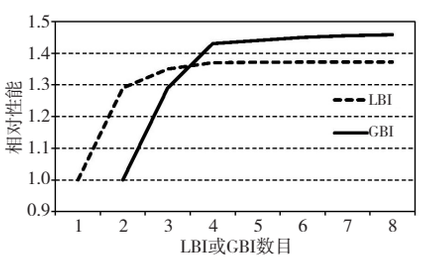

Figure 10 Effect of the number of GBIs and LBIs on performance

图 10 GBI 和 LBI 数量对性能的影响

4.4 硬件开销

对 HSync 进行了功能验证并确认结果正确后,使用 28 nm 工艺库在典型条件下(常温常压(1 V,25 ℃))进行了综合,综合频率达到了 1.56 GHz。综合结果表明,HSync 所占用面积为 401 979.38 μm^2,相比于集中式的 FlatBar 减少了 36%,其中本地栅栏单元约占 66%,全局栅栏单元约占 34%。FlatBar 因为缺乏 FIFO 调度机制,在结点之间需要设置 24 个栅栏实例,而且与结点内的栅栏同步单元没有协同工作,解码器及相应的状态寄存器更多,而 HSync 只需要 12 个全局栅栏实例且逻辑更加简单,因此面积更小。HSync 功耗为 29.46 mW,比 FlatBar 减少了 55.4%,由于 HSync 相比于非层次化 FlatBar 减少了很多重复的解码单元、状态寄存器及控制逻辑,因此降低了功耗。

5 结束语

具有共享内存的多核系统或众核系统中的同步问题需要高效可靠的解决方案。一种简单的方法是利用共享内存实现。然而,事实证明这不是一种高效的方法,且依赖于其他的同步原语。本文提出的设计方案对处理器存储系统没有任何影响,且大大减少了网络中传输的数据包数量;另一方面,由于本地结点内部的同步请求由 LBU 处理,减少了整体同步延迟,提高了系统性能。

栅栏同步使用层次化方式实现的目的在于减少网络内的栅栏同步报文数量,同时减少栅栏同步操作的平均延迟。随着处理器核数的增加,使用 FlatBar 实现的栅栏会将大量同步报文集中到网络的一个结点,造成拥塞等问题,层次化的方式将报文分别在不同结点上进行处理汇集,减少了报文数量,进而降低了对网络的压力,且对于结点内的核可以更快完成同步操作。随着结点数增加及每个结点的核数增加,层次化结构优势更加突出,平均网络延迟也会降低更多。

通过实验结果体现出了层次化硬件栅栏的优势:降低了平均延迟,减少了网络流量,降低了网络开销。本文的实验结果表明,与集中式硬件栅栏 FlatBar 相比,使用层次化栅栏的 HSync 的处理器总执行时间平均减少了 13%,网络流量减少了 74%;与软件栅栏同步方法相比,总执行时间平均减少了 82%。此外,HSync 满足硬件设计的简单性和编程接口的易用性,是一种行之有效的硬件栅栏同步机制。

参考文献:

[1] Scott S L. Synchronization and communication in the multiprocessor[C]//Proc of the 7th International Conference on

Architectural Support for Programming Languages and Operating Systems,1996:26-36.

[2] Beck B,Kasten B,Thakkar S. VLSI assist for a multiprocessor[J]. SIGARCH Computer Architecture News,1987,15(5):10-20.

[3] Hensgen D,Finkel R,Manber U. Two algorithms for barrier synchronization[J]. International Journal of Parallel Programming,1998,17(1):1-17.

[4] Mellor-Crummey J M,Scott M L. Algorithms for scalable synchronization on shared-memory multiprocessors[J]. ACM Transactions on Computer Systems,1991,9(1):21-65.

[5] Luchangco V,Nussbaum D,Shavit N. A hierarchical CLH queue lock[C]//Proc of European Conference on Parallel Processing,2006:801-810.

[6] Yang M,Wieder A,Brandenburg B B. Global real-time semaphore protocols:A survey unified analysis and comparison[C]//Proc of 2015 IEEE Real-Time Systems Symposium,2015:1-19.

[7] Ahn J,Hong S,Yoo S,et al. A scalable processing-in-memory accelerator for parallel graph processing[C]//Proc of the 42nd Annual International Symposium on Computer Architecture,2015:105-117.

[8] Sampson J,Gonzalez R,Collard J F,et al. Exploiting fine-grained data parallelism with chip multiprocessors and fast barriers[C]//Proc of the 39th Annual IEEE/ACM International Symposium on Microarchitecture,2006:235-246.

[9] Lei Z W,Ding H,Xiong H,et al. Design and realization of synchronization technique for FH-π/4-DQPSK communication system[C]//Proc of the 13th IEEE Conference on Industrial Electronics and Applications,2018:2533-2538.

[10] Liang C K,Prvulovic M. Misar:Minimalistic synchronization accelerator with resource overflow management[J]. ACM SIGARCH Computer Architecture News,2015,43(3S):414-426.

[11] Vallejo E,Beivide R,Cristal A,et al. Architectural support for fair reader-writer locking[C]//Proc of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture,2010:275-286.

[12] Marongiu A,Benini L,Kandemir M. Lightweight barrier-based parallelization support for non-cache-coherent MPSoC platforms[C]//Proc of the 2007 International Conference on Compilers Architecture and Synthesis for Embedded Systems,2007:145-149.

[13] Krishnan V,Torrellas J. The need for fast communication in hardware-based speculative chip multiprocessors[C]//Proc of 1999 International Conference on Parallel Architectures and Compolation Techniques,1999:24-33.

[14] Zhu W,Sreedhar V C,Hu Z,et al. Synchronization state buffer:Supporting efficient fine-grain synchronization on many-core architectures[C]//Proc of the 34th Annual International Symposium on Computer Architecture,2007:35-45.

[15] Zeng K W,Ning M N,Wang Y H,et al. Hierarchical clustering with hard-batch triplet loss for person re-identification[C]//Proc of the IEEE/CVF Conference on Computer Vision and Pattern Recognition,2020:13657-13665.

[16] Chen Xiao-wen. Dual channel fast barrier synchronization mechanism based on cooperative communication[EB/OL]. [2017-05-12]. https://www.doc88.com/p-5307498419896.html. (in Chinese)

[17] Shang S,Hwang K. Distributed hardwired barrier synchronization for scalable multiprocessor clusters[J]. IEEE Transactions on Parallel and Distributed Systems,1995,6(6),591-605.

[18] Villa O,Palermo G,Silvano C. Efficiency and scalability of barrier synchronization on NoC based many-core architectures[C]//Proc of the 2008 International Conference on Compilers Architectures and Synthesis for Embedded Systems,2008:81-90.

[19] Hsu W T Y,Yew P C. An effective synchronization network for hot-spot accesses[J]. ACM Transactions on Computer Systems,1992,10(3):167-189.

[20] Monchiero M,Palermo G,Silvano C,et al. An efficient synchronization technique for multiprocessor systems on-chip[C]//Proc of the 2005 Workshop on Memory Performance:Dealing with Applications Systems and Architecture,2005:33-40.

[21] Giannoula C, Vijaykumar N, Papadopoulou N,et al. SynCron: Efficient synchronization support for near-data-processing architectures[C]//Proc of 2021 IEEE International Symposium on High-Performance Computer Architecture,2021:263-276.

[22] Hetland C,Tziantzioulis G,Suchy B,et al. Paths to fast barrier synchronization on the node[C]//Proc of the 28th International Symposium on High-Performance Parallel and Distributed Computing,2019:109-120.

[23] Liu Chang, Guo Yang. Instruction set verification method for high performance DSP based on coverage-driven[J]. Computer Engineering,2014,40(6):317-320. (in Chinese)

[24] Chen Hai-yan, Guo Yang, Chen Ji-hua. Computer aided design and verification practice of integrated circuit[M]. Changsha: National University of Defense Technology Press,2010. (in Chinese)

附中文参考文献:

[16] 陈小文. 基于协同通信的双通道快速栅栏同步机制[EB/OL]. [2017-05-12]. https://www.doc88.com/p-5307498419896.html.

[23] 刘畅,郭阳. 基于覆盖率驱动的高性能DSP指令集验证方法[J]. 计算机工程,2014,40(6):317-320.

[24] 陈海燕,郭阳,陈吉华. 集成电路计算机辅助设计与验证实践[M]. 长沙:国防科技大学出版社,2010.

作者简介:

臧照虎(1995-),男,河南淮滨人,硕士生,研究方向为微处理器设计与验证。**E-mail**:zangzhaohu@gmail.com

ZANG Zhao-hu, born in 1995, MS candidate, his research interest includes microprocessor design & verification.