计算机科学
COMPUTER SCIENCE

一种面向申威 26010 处理器的分布式传递锁机制

李明亮，庞建民，岳峰

引用本文

李明亮，庞建民，岳峰. 一种面向申威 26010 处理器的分布式传递锁机制[J]. 计算机科学, 2022, 49(10): 52-58.

LI Ming-liang, PANG Jian-min, YUE Feng. Distributed Lock with Inter-core Passing for SW26010 Processor[J]. Computer Science, 2022, 49(10): 52-58.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210800091

一种面向申威26010处理器的分布式传递锁机制

李明亮　庞建民　岳　峰
数学工程与先进计算国家重点实验室(信息工程大学)　郑州 450000
(lmliang_only@163.com)

摘　要　在并行程序中,互斥锁通常被用来避免访问共享资源时发生冲突。申威26010处理器是"神威·太湖之光"超级计算机采用的异构众核处理器,众核之间并无硬件互斥锁机制。其开发人员基于原子操作实现了一种软件互斥锁,但是该软件锁在激烈锁竞争情况下会产生大量的锁操作开销,影响了并行程序的性能。针对这一问题,提出了一种分布式传递锁机制HDT-LOCK。首先,提出并实现了基于众核上便签存储器和主存的混合分布锁来避免访存拥塞;其次,设计了基于寄存器通信和单指令多数据指令(Single-instruction Multiple-data Instruction)的锁传递机制,以进一步提高HDT-LOCK机制的吞吐量。实验结果表明,与原锁机制相比,所提HDT-LOCK机制避免了访存拥塞,并且可扩展性更佳。此外,锁传递机制使HDT-LOCK的吞吐量提升最高可达5.6倍。

关键词:申威26010处理器;混合分布锁;锁传递;单指令多数据指令;寄存器通信
中图法分类号　TP319

Distributed Lock with Inter-core Passing for SW26010 Processor

LI Ming-liang,PANG Jian-min and YUE Feng
State Key Laboratory of Mathematical Engineering and Advanced Computing,PLA Information Engineering University,Zhengzhou 450000,China

Abstract　In parallel programs,a mutual exclusive lock is often used to avoid conflict when accessing shared resources. The SW26010 processor,which is deployed on the Sunway TaihuLight supercomputer,is a heterogeneous many-core processor and there is no hardware lock mechanism for the co-processing cores. Developers have developed a software lock mechanism based on atomic instructions,but the software lock will lead to significant overhead and affect the performance of parallel programs. To solve this issue,the HDT-LOCK designed as distributed lock mechanism with inter-core passing is proposed. Firstly,the hybrid distributed lock is proposed and implemented based on scratchpad memory on co-processing cores to mitigate memory congestion. Furthermore,the inter-core passing mechanism using register communication and the single-instruction multiple-data instruction is developed to improve the throughput of HDT-LOCK. Experimental results show that the proposed HDT-LOCK mechanism mitigates memory congestion,and has better scalability. In addition,the lock passing mechanism improves HDT-LOCK throughput up to 5.6X.

Keywords　SW26010 processor,Hybrid distributed lock,Inter-core passing,Single-instruction multiple-data instruction,Register communication

1　引言

在并行程序中,互斥锁机制是保证并发线程在访问共享资源时不发生冲突的重要机制。若线程在访问共享资源时发生"交错",将产生不可预知的程序运行结果。

随着申威处理器的日渐成熟,其处理器功能和软件生态也在不断完善。众多研究人员利用其独特的架构进行了诸多程序移植与优化工作[1-2]。除其本身具有的众核加速编程模型外,申威处理器还支持包括MPI和OpenACC在内的多种并行设计语言和模型。并行编程标准OpenMP以及OpenACC等中的一些线程控制机制也需要基于互斥锁机制实现,如OpenACC标准中的critical编译指示[3]。该临界区控制通常基于互斥锁机制实现[4]。在进入临界区之前,线程进行加锁操作,以保证其他线程无法同时获得锁并进入临界区。在代码运行完毕并退出临界区时,线程进行解锁操作,从而使得下一个线程可以获取锁并执行临界区代码。

互斥锁机制可以基于硬件或者软件实现。典型的硬件锁机制,如总线锁机制(Bus Locking),需要底层平台提供相应的硬件特性。软件锁机制大多是自旋锁(Spin Lock)[5],其特点是线程在加锁不成功时,会不断尝试获取锁或者对锁变量进行忙等待(Busy-waiting),直到加锁成功。简单的自旋锁主要是基于"比较并交换"原子指令的CAS(Compare And

到稿日期:2021-08-11　返修日期:2022-01-07
基金项目:国家自然科学基金(61472447,61802433,61802435)
This work was supported by the National Natural Science Foundation of China(61472447,61802433,61802435).
通信作者:庞建民(jianmin_pang@126.com)

Swap)原子锁和基于“测试并写入”原子指令的 TAS(Test And Set)原子锁[6]。

申威处理器没有提供众核间硬件锁机制，也没有提供 CAS 和 TAS 原子指令。因此，在基于申威 26010 处理器的申威 OpenACC 语言实现中，开发人员为众核提供了一种基于“读并加”原子指令的软件自旋锁机制，本文将其称之为“FAA(Fetch And Add)锁”。FAA 锁机制的实现原理与 TAS 原子锁类似。

这些自旋锁面临的问题是，若存在激烈的锁竞争，大量的线程因无法获取锁而在锁变量上不断自旋，从而导致大量的访存开销。在申威众核架构下，FAA 锁机制中的锁变量位于主存中且众核数目最多可达 64 个。实验发现，FAA 锁机制会导致访存拥塞，且锁操作开销随线程数目的增加而迅速增大，严重影响了并行程序的运行效率。尽管研究人员已经提出了众多方法来优化自旋锁的开销问题[7-9]，但是目前仍缺乏适合申威架构的有效手段。

针对申威 26010 处理器上原有锁机制面临的访存拥塞问题，本文提出一种分布式传递锁机制。该机制利用众核上的便签存储器和主存实现了锁机制的混合分布，避免了访存拥塞；在此基础上，设计了基于寄存器通信和 SIMD(Single Instruction Multiple Data)指令的锁传递机制，进一步提高了锁机制效率。

2 申威 26010 处理器架构

申威 26010 处理器[10]采用异构众核架构，每个处理器包含 4 个核组(Core Group，CG)。其中，每个核组上配置有 1 个运算控制核心(Management Processing Element，MPE)和 64 个运算核心(Computing Processing Elements，CPE)。本文将 MPE 和 CPE 分别称为主核和从核。申威 26010 处理器单个核组的结构如图 1 所示。

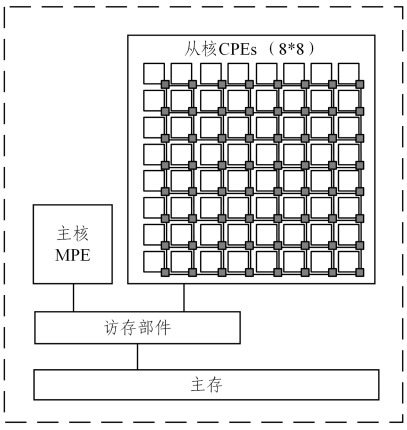


图 1 申威 26010 处理器核组结构图

Fig. 1 Architecture of one core group of SW26010

主存作为核组中的全局存储空间，可以同时被主核和 64 个从核访问。从核对主存上的数据读写被称为全局读写，表示为 GLD/GST。值得注意的是，所有的从核在访问主存时都需要通过同一个访存器件进行。因此，在高并发情况下，所有从核在访问主存时将竞争使用访主存带宽。

每个从核都配备有私有的高速便签存储器(ScratchPad Memory，SPM)作为局部存储空间。从核访问 SPM 时，带宽大、延时低。每个从核只能访问其私有的 SPM，无法访问其他从核的 SPM。因此，从核间共享数据只能通过主存或者从核间通信机制进行。主核既可以访问主存，也可以直接访问从核的 SPM。

3 FAA 锁机制原理及存在的问题

如前文所述，从核没有提供 CAS 和 TAS 原子指令。受限于硬件结构，申威提供了基于“读并加”原子指令的从核间锁机制 FAA 锁。下文将给出 FAA 锁机制的实现原理并分析其存在的问题。

从核上可以使用的“读并加”原子指令有 faaw 和 faal 两种，分别应用于主存上 32 位和 64 位整数。由于两种指令功能相同，下面主要以 faaw 指令为例进行阐述。“读并加”原子指令的功能是原子地将主存中的整数变量的值读取到从核寄存器中，并在主存中将该变量的值加 1。算法 1 以伪代码的形式给出了指令的功能。

算法 1 faaw 原子指令功能

```
输入：(global_var)
输出：(local_value)
1. /* faaw 原子指令功能示例 */
2. 读取数据到本地：local_value ←global_var
3. 全局值加 1：global_var ←global_var+1
4. 指令返回原值：return local_value
```

FAA 锁机制基于该原子指令实现。其锁机制的原理是，在主存中设置两个锁变量来保存互斥锁状态，并使用原子指令对其进行操作来实现加锁和解锁操作。

算法 2 和算法 3 以 faaw 指令为例，以伪代码的形式给出了 FAA 锁的加锁和解锁工作原理。从核通过 faaw 指令对 *global_req_n* 进行操作来进行加锁请求。*global_req_n* 代表目前加锁的请求号，初始化为 0；*global_res_n* 代表目前可以获得锁资源的请求号，初始化为 1。从核进行 faaw 操作后会获得请求号 *local_req*，并对全局锁变量 *global_req_n* 加 1。由于该过程由一条原子指令完成，因此多个从核之间并不会产生冲突。当 *local_req* 与 *global_res_n* 相等时，加锁成功并开始执行临界区代码。而当两者不相等时，表示当前有其他从核正在使用锁，本从核将对 *global_res_n* 进行忙等待直到两者相等，即不断读取主存中 *global_res_n* 的值并判断其与 *local_req* 是否相等。当占有锁资源的从核进行解锁操作时，通过 faaw 指令操作 *global_res_n* 释放锁资源。此时进行了加锁请求的下一个从核将成功获取锁资源。

算法 2 基于 faaw 原子指令的加锁操作原理

```
输入：(global_req_n，global_res_n)
1. /* 加锁操作原理 */
2. 取加锁号：
    local_req ←faaw( global_req_n)
3. 忙等待直到获取锁：
while global_res_n ≠local_req do
        忙等待
```

算法 3 基于 faaw 原子指令的解锁操作原理

```
输入：(global_req_n，global_res_n)
1. /* 解锁操作原理 */
2. 释放锁：faaw( global_res_n )
```

FAA 锁机制也面临经典自旋锁带来的访存拥塞问题。

用T_A表示原子操作开销，用T_W表示忙等待开销，则由算法2可知，FAA锁机制的加锁开销T可以表示为：

$$T=T_A+T_W \tag{1}$$

实验发现，随着参与锁竞争的从核数目的增加，T_W急剧增加，从而导致加锁开销也急剧增大。

事实上，T_W急剧增加是访存拥塞导致的[6]。分析算法2可知，从核在尝试加锁失败后会对主存中锁变量*global_res_n*进行忙等待。当存在激烈的锁冲突时，大多数从核无法获取锁资源。这些从核会不断发起访存请求以获取锁状态。在申威26010处理器中，同一个核组内的从核通过同一个访存部件对主存进行读写。访存部件为GLD/GST分别设置了队列缓冲，短时间内大量的访存请求会被置入缓冲中进行排队处理。因此，众多从核忙等待时产生的大量访存请求会导致访存拥塞。

访存拥塞使得从核单次加锁的开销陡然增加，从而导致FAA锁机制的吞吐量迅速下降，第5.2节中的测试数据也证明了这一点。因此，需要设计新的锁机制来避免这一问题。

4 分布式传递锁 HDT-LOCK

为了解决FAA锁机制面临的访存拥塞问题，本文设计并实现了分布式传递锁机制HDT-LOCK。在本节中，我们将其分为混合分布锁机制和锁传递机制两个部分进行详细阐述。混合分布锁机制是HDT-LOCK机制的基础，因此我们将首先讨论该机制的设计。锁传递机制的设计将在4.2节进行阐述。

4.1 混合分布锁机制

由第3节的分析可知，FAA锁机制主要存在以下两个方面的不足：

(1)全部锁变量都位于主存中，从核访问主存需要大量的时间开销。

(2)当从核进行加锁请求后，需要对主存中的变量进行忙等待。当参与锁竞争的从核数目较多时，将导致内存拥塞，继而引起锁操作开销急剧增加。

申威26010处理器中从核进行GLD/GST访问主存时，带宽受限且访存消耗时钟周期久；而在访问从核私有SPM时，带宽较大且访存时钟周期极短。此外，主核可以直接访问从核SPM空间，且访存性能约为GLD/GST的3倍[11]。

基于以上分析，本文提出了一种混合分布锁机制，其原理如图2所示。锁机制同时利用主核和从核进行锁资源的管理和使用。从核通过GST设置主存中的对应标志位来发送加锁请求，然后对位于SPM的私有标志位进行忙等待。主核负责查询和处理从核的加锁请求，并通过直接访问SPM来通知

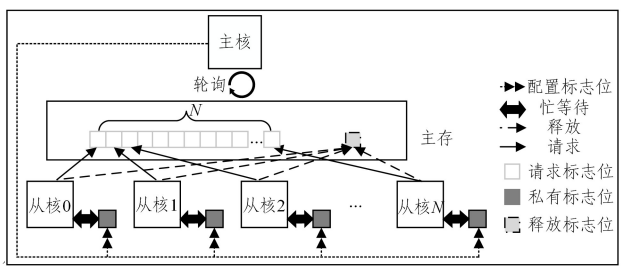


图2 混合分布锁机制原理

Fig. 2 Principle of hybrid distributed lock

机制中设置的标志位主要分为以下3种：

(1)请求标志位：位于主存中。每个从核线程都在主存中设置有对应的标志位。从核通过GST设置请求标志位来进行加锁请求。

(2)私有标志位：位于每个从核的SPM中。私有标志位用于标识本从核的加锁结果。从核在设置请求标志位的加锁请求后，会在私有标志位上进行自旋。当主核调度锁资源为本从核使用后，会修改对应从核的私有标志位，则该从核将退出自旋，进入临界区。

(3)释放标志位：位于主存中。释放标志位用于指示对应锁是否处于空闲状态。标志位初始化为空闲状态。当主核分配锁资源给从核时，其被主核设置为占用状态，而从核在使用完毕后将其重新修改为空闲状态。主核通过读取释放标志位的状态来决定是否将锁资源分配给下一个从核。

与FAA锁机制相比，本文提出的锁机制主要有两个优点。1)分布式的机制设计避免了访存拥塞。与FAA锁机制不同，锁机制的变量分布于主存和SPM中。从核在发送加锁请求后，转而对位于SPM中的私有标志位进行忙等待。由于从核不再对主存中的锁变量进行大量的GLD操作，也就避免了访存拥塞。2)利用了主从核间不对称的访存带宽。在从核加锁成功时，由主核直接修改从核SPM中的标志位，其开销显著优于从核通过GLD操作读取主存中的锁变量。

混合分布锁机制的工作流程如图3所示。在进行加锁操作时，从核首先设置对应的请求标志位发送加锁请求，然后进入对私有标志位的忙等待。主核将资源分配给该从核时，会通过直接访问SPM来将该从核的私有标志位设置为可用状态。从核在检测到标志位发生改变后，即认为获得了锁资源，退出忙等待并进入临界区执行代码。代码执行完毕退出临界区时，从核进行解锁操作，即设置释放标志位为空闲状态，结束本次临界区操作并继续执行后续代码。

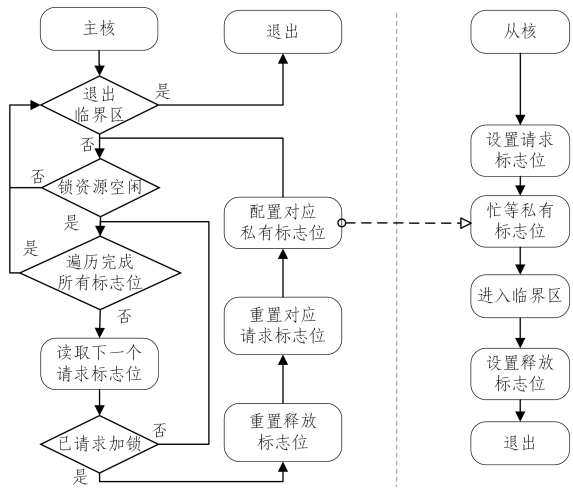


图3 混合分布锁机制工作流程

Fig. 3 Workflow of hybrid distributed lock

若某一锁资源已经被上一从核线程使用完毕，处于空闲状态，则主核线程会查询下一个从核是否进行了加锁操作。若是，则主核线程将对应从核的请求标志位恢复为初始状态，并设置其私有标志位，使其进入临界区。

若当前锁正在被占用，主核会再次读取需要处理的锁状态。当只有一个锁资源需要调度时，主核会再次查询该锁

资源的状态。当有多个锁需要同时处理时，主核会对多个锁的状态进行轮询并依次处理。

4.2 锁传递机制

混合分布锁机制解决了访存拥塞的问题，然而通过开销分析，我们发现其还存在冗余通信开销。本节将在混合分布锁机制的基础上引入锁传递机制。一次锁操作开销由加锁开销和解锁开销组成。令 T_s 表示设置标志位开销，T_W 表示忙等待开销，T_r 表示释放标志位开销，则混合分布锁机制的单次操作开销可以表示为：

$$T = T_s + T_W + T_r \tag{2}$$

对上述开销进行分析可以发现，当某一从核释放锁资源并由主核调度给下一从核使用时，会带来释放标志位开销和主核设置私有标志位开销。其中，从核释放锁资源需要进行 GST 操作来设置主存中的释放标志位，所需开销较大。在激烈锁竞争情况下，当一个从核占用锁资源时，同时也会有多个从核处于等待状态。如果锁资源可以直接在从核间传递，则无须主核和从核间的通信操作就可以进一步降低开销和提升锁机制的吞吐量。

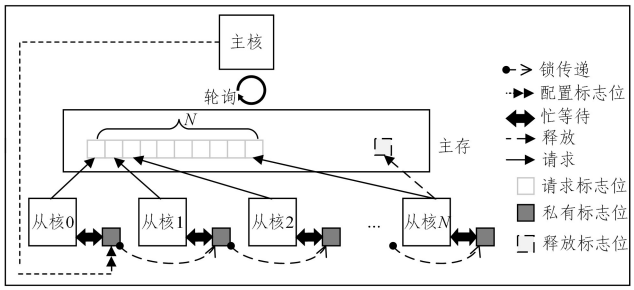


图 4 HDT-LOCK 机制原理

Fig. 4 Principle of HDT-LOCK

申威 26010 处理器提供从核间寄存器通信机制。从核以 8 * 8 的阵列组织，位于同一行或者同一列的从核可以通过寄存器通信机制传递数据。

基于以上分析，本文在 HD-LOCK 机制的基础上提出了一种基于 SIMD 指令和寄存器通信机制的分布式传递锁机制 HDT-LOCK，以减少激烈锁竞争情况下主核和从核间的冗余通信开销，进一步提升锁机制的吞吐量。图 4 给出了 HDT-LOCK 的工作原理，该机制的核心是，当一行从核都需要获取锁资源时，通过利用寄存器通信机制使得锁资源在从核间传递。每一个从核在其执行结束后释放锁资源时，不再直接设置主存中的释放标志位，将锁资源交还给主核分配，而是通过寄存器通信机制将锁资源传递给下一个从核。该方法避免了锁资源传递时主从核间的冗余通信开销，即从核设置释放标志位和主核设置私有标志位的开销。值得注意的是，实验结果表明，寄存器通信开销仅为锁资源释放再分配开销的几十分之一。

分布式传递锁机制使用申威 26010 处理器提供的 SIMD 指令来降低主核进行标志位判断时的开销。主核可以使用 SIMD 指令同时判断一行 8 个从核的标志位状态，当所有请求标志位均为有效时，SIMD 指令返回值为 1；当标志位中有任何一个无效时，SIMD 指令返回值为 0。使用分布式传递锁机制中主核和从核的工作流程图如图 5、图 6 所示。

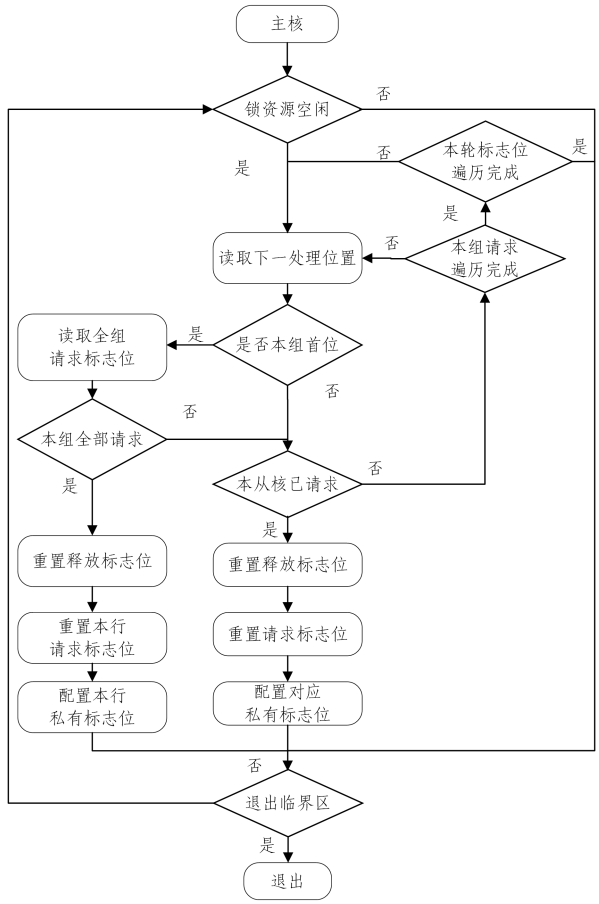


图 5 HDT-LOCK 主核工作流程

Fig. 5 Workflow of MPE in HDT-LOCK

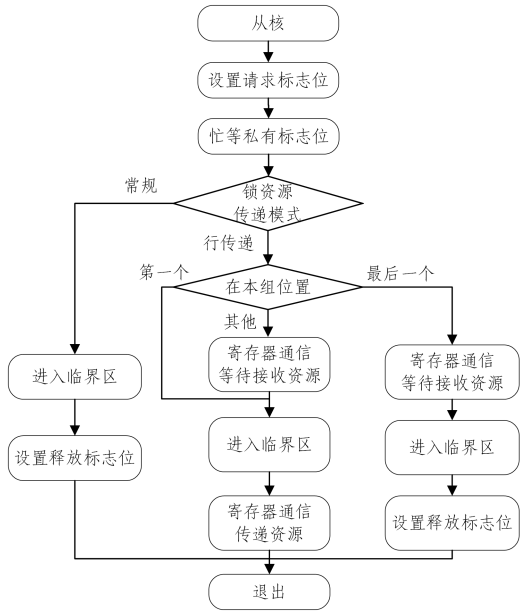


图 6 HDT-LOCK 从核工作流程

Fig. 6 Workflow of CPE in HDT-LOCK

在进入临界区时，从核向主核发送请求。主核在处理从核请求时，首先通过 SIMD 指令判断当前组的从核是否都发送了请求。若是，则将锁资源分配给本组第一个从核，并通过配置其他从核的标志位使其进入同行传递模式。而后，主核

进入忙等待状态，直到本组所有的从核退出临界区，并设置释放标志位。

如果本组中有任何一个从核没有发送请求，则主核进入常规处理模式，依次处理本组从核的请求。

从核在向主核发送请求后，进入忙等待模式。通过判断私有标志位的值，来判断当前应该采用哪种工作模式。当工作模式为同行传递时，根据自身在本行的位置，采用不同的工作方式。最后一个从核在结束临界区操作后，向主核发送释放资源请求。

5 实验测试

为了评估本文提出的分布式传递锁机制的性能，本节对其进行实验测试。我们在申威 26010 处理器上分别实现了不包含锁传递机制的混合分布锁机制 HD-LOCK 和分布式传递锁机制 HDT-LOCK，并将其封装为接口函数以方便调用。本节主要通过实验设置回答以下 3 个具体问题。

(1)与原锁机制相比，本文提出的锁机制性能如何？

(2)不同从核数目对互斥锁机制性能的影响如何？

(3)随着同时处理的锁数目的增加，不同互斥锁机制的可扩展性如何？

5.1 测试环境和测试程序

实验选取的测试平台为“神威·太湖之光”超级计算机，采用的处理器为申威 26010 处理器。由于互斥锁机制是核组内从核间的作用机制，因此实验只使用其中的一个核组，包括 1 个主核和 64 个从核。

衡量锁机制性能主要使用吞吐量这一指标。实验采用 EPCC OpenACC 测试集[12]中的 syncbench 程序测试并计算得出不同锁机制的吞吐量。我们使用 syncbench 程序测得的 critical 编译指示的开销计算得出平均锁操作开销。critical 编译指示用于控制线程进入临界区，保证同一时间只能有一个进程执行临界区代码。在申威 OpenACC 实现中，此编译指示实现为在进入临界区前后分别调用 FAA 锁进行加锁和解锁操作，因此，根据这一编译指示的开销可以计算得出锁机制的吞吐量。测试程序的输出为所有从核线程完成临界区操作的总开销，而同一时间只能有一个线程获取锁，因此锁机制的吞吐量可以由式(3)计算得到：

$$\text{吞吐量}=\frac{N*F}{Time_{critical}} \tag{3}$$

其中，N 表示临界区中的从核线程数目，$Time_{critical}$ 表示由测试集测得的 N 次加锁和解锁操作节拍数之和，F 表示处理器频率。通过将测试程序中的锁机制替换为本文提出的锁机制，我们就可以得到不同锁机制的吞吐量数据。

5.2 性能比较

本组实验的目的是回答前两个问题，即本文提出的锁机制和原锁机制的性能比较以及不同从核线程数目对锁机制性能的影响。图 7 给出了不同从核线程数目下，FAA 锁(原锁机制)、混合分布锁 HD-LOCK 和分布式传递锁 HDT-LOCK 的性能对比。实验中，参与锁竞争的从核线程数目为 N_s ($N_s=2,4,8,16,32,64$)。

从图 7 中可以看出，在从核线程数目为 2 时，3 种锁机制的吞吐量最接近。随着从核数目的增加，FAA 锁机制的吞吐量急剧下降。在从核线程数为 64 时，FAA 锁机制的吞吐量仅为 0.14 Mops/s。这一现象也与前文的分析相符；而 HD-LOCK 机制的吞吐量虽略有下降，但始终在 1.45 $ Mops/s 左右；HDT-LOCK 机制的吞吐量则随从核数目的增加迅速提升，在从核数目为 16 时达到最大，而后又随着从核数目的增加缓慢降低。相比原锁机制，HD-LOCK 机制和 HDT-LOCK 机制的性能最高可提升 10.5 倍和 56.9 倍。这一实验结果表明，分布式锁机制避免了从核线程在主存锁变量上自旋导致的访存拥塞。仅在从核数目为 2 时，原锁机制与 HD-LOCK 机制的性能接近。然而，考虑到并行加速程序通常会利用尽可能多的加速线程来获得更高的执行效率，从核数目小于 2 的情况极少出现。

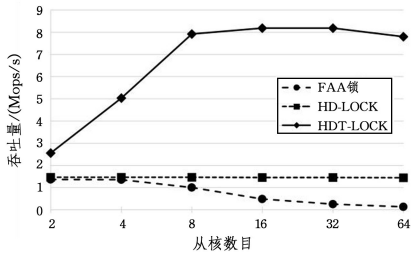


图 7 3 种锁机制吞吐量对比

Fig. 7 Comparison of throughput of three lock mechanisms

分析 HDT-LOCK 机制的性能可以看出，其性能明显优于其他两种锁机制。与 HD-LOCK 相比，锁传递机制使 HDT-LOCK 的吞吐量提升最高可提升 5.6 倍。在 HDT-LOCK 机制中，锁资源在从核间进行了传递，且传递开销远远小于锁资源释放并再次赋予下一个从核的开销。因此，得益于更低的锁操作开销，HDT-LOCK 的性能明显优于 HD-LOCK。由于从核间寄存器通信的开销极低，锁传递机制的开销主要取决于一组从核进行主存读写的开销。而在锁传递机制的设计中，其一组从核访问主存的时间开销只包含发送请求和本组最后一个从核释放锁资源两个操作带来的开销，而这两个操作的时间几乎不随从核数目的变化而变化。因此，一组从核进行加锁解锁操作的开销被平均分摊，且参与分摊的从核数目越多，每个从核进行锁操作需要的开销也就越低。然而，这种开销的分担取决于能够参与传递的从核数目，由于申威 26010 处理器拓扑结构的限制，只有同行或同列的 8 个从核间可以直接进行寄存器通信。在当前的 HDT-LOCK 设计中，也只判断并处理同行从核间的锁传递。因此，当锁机制中总从核个数大于一行从核的个数时，平均锁操作开销随着需要处理从核请求数目的增加而增加，吞吐量也随之略微下降。

5.3 可扩展性测试

为了进一步研究本文提出的锁机制的性能，本组实验对 3 种锁机制的可扩展性进行了测试，即测量同时存在多个互斥锁的情况下，几种互斥锁机制的性能变化。本节实验构造了多个从核线程分组竞争使用多个锁的测试用例。在测试中对从核进行分组，分别竞争获取 $N(N=2,3,4)$ 个锁。因为

每个核组只有 64 个从核，所以在锁数目 $N=4$ 时，每组最多只能有 16 个从核参与锁竞争。实验使用单次锁操作平均开销的变化来衡量锁机制的可扩展性能。由于同时存在多个锁，锁操作平均开销会随锁数目的增加而增加。开销增加越少，锁机制的可扩展性越好。单次锁操作的平均开销可以由式(4)计算得到。

$$\text{平均开销}=\frac{1}{\text{吞吐量}} \tag{4}$$

实验结果如图 8 所示，锁数目 $N=2$ 时的开销被归一化为 1。从实验结果可以看出，随着锁数目的增加，3 种锁机制的平均开销也相应增加。其中，原锁机制的开销增幅最大，HD-LOCK 机制次之，HDT-LOCK 机制增幅最小。以 $N=2$ 为基准，锁的数目 N 每增加 1，原锁机制的平均开销分别为 1.47 和 1.87。从核同时对多个锁的竞争访问，导致访存拥塞更加严重，每次锁操作的平均开销也更大。

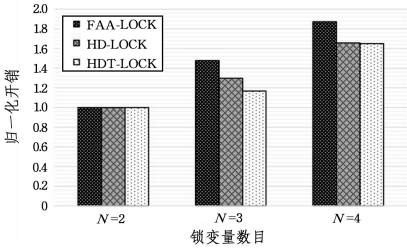


图 8 锁机制的可扩展性对比

Fig. 8 Comparison of scalability of lock mechanisms

虽然两种分布式锁机制的开销均随锁数目的增加而增加，但是其增幅明显小于原锁机制。锁变量的数目分别增加至 3 和 4 时，HD-LOCK 机制的开销分别为 1.29 和 1.65，HDT-LOCK 机制的开销分别为 1.16 和 1.64。开销增加主要来源于主核需要同时处理的加锁请求增加。由于主核串行处理多个锁的加锁请求，在主核处理能力不变的情况下，锁数目增加必然导致平均开销增加。

通过以上分析可以得出，相比原锁机制，本文提出的 HDT-LOCK 机制可扩展性更佳，HD-LOCK 机制次之。

6 相关工作

锁机制是确保并行程序访问共享资源时不发生冲突的重要手段，但是目前针对申威处理器众核间锁机制的研究还很少。FAA 锁是一种自旋锁。在激烈的锁竞争的情况下，自旋锁的性能会变得很差[13]。为了解决锁冲突导致性能变差的问题，研究人员展开了大量的研究。指数退避算法[14]是一种简单的解决方法，但是存在锁资源利用率不高的问题。

队列锁是针对锁冲突问题的另一类解决方案。MCS 锁和 CLH 锁[15]是两种具有代表性的队列锁。队列锁机制在内存中维护一个队列。在进行加锁时，线程将自身添加到队列的末尾。当线程释放锁资源时，通过查询队列，将锁资源传递给队列中的下一个线程。队列锁机制较好地解决了锁竞争问题，且有着比指数避让算法更好的性能[13]。但是，队列锁机制普遍基于“比较并交换”原子指令来进行队列操作，而申威处理器上并没有为从核提供该类原子指令。

事实上，锁机制在分布式系统中也被广泛使用，并且按照对锁资源的管理方式大致可以分为中心化锁[16]和去中心化锁[17]两种。中心化锁通常采用一个节点作为管理节点，管理和分配锁资源，而其他节点在获取锁资源时需要与管理节点通信。因此，单个节点的处理能力和节点间通信开销成为制约中心化锁吞吐量提升的瓶颈。为解决中心化锁机制的问题，去中心化的方法采用分布式的锁管理方法，通过机制设计使众多节点共同参与锁机制的管理。但是，受限于申威处理器的硬件特性，直接将去中心化锁应用到申威处理器中会导致相应从核被占用，无法进行加速计算[18]。从核阵列中的部分从核无法参与计算，也破坏了众核加速编程接口的一致性。

本文提出的 HDT-LOCK 机制结合了中心化锁和去中心化锁两种机制的优点，不但突破了中心化锁的瓶颈，而且不占用从核作为管理节点，机制不改变原有的众核加速编程接口。通过直接替换申威并行编程模型中的锁机制库函数使用，本文提出的锁机制即可快速应用到众多利用到锁的众核加速程序中。

结束语 本文面向申威 26010 处理器提出了一种分布式传递锁机制 HDT-LOCK，利用众核便签存储器和主存两种不同层次的内存实现了混合分布锁 HD-LOCK，并在此基础上提出了锁传递机制，利用申威处理器提供的 SIMD 指令和从核间寄存器通信手段降低了机制中的冗余通信开销，进一步提升了吞吐量。实验结果表明，与原锁机制相比，本文提出的 HDT-LOCK 机制避免了访存拥塞，获得了显著的性能提升，可扩展性更佳；同时，与 HD-LOCK 相比，锁传递机制使 HDT-LOCK 的吞吐量最高可提升 5.6 倍。

参 考 文 献

[1] ZHU Y, PANG J M, XU J L, et al. Adaptive Tiling Size Algorithm for 3D Stencil Computation on SW26010 Many-core Processor[J]. Computer Science, 2021, 48(6): 10-18.

[2] TAO X H, PANG J M, GAO W, et al. Performance Optimization of FT Program Based on SW26010 Processor[J]. Computer Science, 2019, 46(4): 321-328.

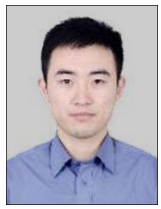
[3] WIENKE S, SPRINGER P, TERBOVEN C, et al. OpenACC—first experiences with real-world applications[C]// European Conference on Parallel Processing. Berlin: Springer, 2012: 859-870.

[4] DALESSANDRO L, DICE D, SCOTT M, et al. Transactional mutex locks[C]// European Conference on Parallel Processing. Berlin: Springer, 2010: 2-13.

[5] ALFRANSEDER M, DEUBZER M, JUSTUS B, et al. An efficient spin-lock based multi-core resource sharing protocol[C]// 2014 IEEE 33rd International Performance Computing and Communications Conference (IPCCC). Piscataway: IEEE Press, 2014: 1-7.

[6] DUAN X H. Optimization of Molecular Dynamics Algorithms Based on the Sunway TaihuLight Supercomputer[D]. Jinan:

Shandong University, 2020.
[7] CHABBI M, FAGAN M, MELLOR-CRUMMEY J. High performance locks for multi-level NUMA systems[J]. ACM SIGPLAN Notices, 2015, 50(8): 215-226.
[8] DICE D. Malthusian locks [C]// Proceedings of the 12th European Conference on Computer Systems (EuroSys'17). New York: ACM, 2017: 314-327.
[9] DICE D, MARATHE V J, SHAVIT N. Lock cohorting: A general technique for designing NUMA locks[J]. ACM Transactions on Parallel Computing (TOPC), 2015, 1(2): 1-42.
[10] FU H, LIAO J, YANG J, et al. The Sunway TaihuLight supercomputer: system and applications [J]. Science China-Information Sciences, 2016, 59(7): 1-16.
[11] CHEN D X, LIU X. Parallel programming and optimization of Sunway TaihuLight[M]. Wuxi: National Parallel Computer Engineering Technology Research Center, 2017.
[12] EPCC. EPCC OpenACC Benchmarks [EB/OL]. (2013-09-23) [2021-08-10]. https://github.com/EPCCed/epcc-openacc-benchmarks.
[13] ANDERSON T E. The performance of spin lock alternatives for shared-memory multiprocessors[J]. IEEE Transactions on Parallel and Distributed Systems, 1990, 1(1): 6-16.
[14] KWAK B J, SONG N O, MILLER L E. Performance analysis of exponential backoff[J]. IEEE/ACM Transactions on Networking, 2005, 13(2): 343-355.
[15] CRAIG T. Building FIFO and priorityqueuing spin locks from atomic swap: Technical Report TR 93-02-02[R]. Seattle: Department of Computer Science, University of Washington, 1993.
[16] GOPALAKRISHNA K, LU S, ZHANG Z, et al. Untangling cluster management with Helix[C]// Proceedings of the Third ACM Symposium on Cloud Computing. New York: ACM, 2012: 1-13.
[17] FRANCE-PILLOIS M, MARTIN J, ROUSSEAU F. Implementation and evaluation of a hardware decentralized synchronization lock for MPSoCs[C]// 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). Piscataway: IEEE Press, 2020: 1112-1121.
[18] TANG X, ZHAI J, QIAN X, et al. plock: A fast lock for architectures with explicit inter-core message passing[C]// Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2019: 765-778.



LI Ming-liang, born in 1991, Ph.D. His main research interests include high-performance computing and binary translation.

PANG Jian-min, born in 1964, Ph.D, professor, Ph.D supervisor, is a member of China Computer Federation. His main research interests include high-performance computing and information security.

(责任编辑:杨雪敏)