基于硬件锁的多线程同步设计和实现

李春江 唐 滔 杨灿群
(国防科学技术大学计算机学院 长沙 410073)

摘 要 硬件锁用简单的取数指令实现"取并加一"或"取并减一"的原子操作。首先介绍了通用多核多线程FT处理器实现的硬件锁机制,并和软件锁机制进行了比较,之后介绍了使用硬件锁机制实现多线程同步的方法,然后在GNU OpenMP运行库中设计并实现了利用硬件锁的多线程同步机制,最后采用典型OpenMP测试程序对使用硬件锁和使用软件锁的同步操作性能进行了评估和分析。

关键词 硬件锁,同步,FT处理器,GNU OpenMP运行库
中图法分类号 TP314 **文献标识码** A

Design and Implementation of Synchronization for Multithreading Based on Hardware Locks

LI Chun-jiang TANG Tao YANG Can-qun
(School of Computer Science, National University of Defense Technology, Changsha 410073, China)

Abstract Hardware locks provide the atomic operation mechanism of "load and add one" or "load and subtract one" by simple load instructions. Firstly, we introduced the implementation of hardware locks in general purpose multi-core multi-threaded FT processors, and compared with the software lock mechanism, then described the method for synchronization of multi-threading using hardware locks. In the GNU OpenMP runtime library, we designed and implemented the synchronization mechanism based on hardware locks for FT processors. Finally, we evaluated the performance of typical OpenMP programs using hardware locks versus using software locks and gave some valuable analysis.

Keywords Hardware locks, Synchronization, FT processor, GNU OpenMP runtimes

1 引言

基于锁的同步机制是多线程并行应用中同步操作的主要实现机制。多核多线程通用处理器一般都支持"比较并交换"原子操作指令,如UltraSPARC Architecture 2007体系结构中定义的"casa"和"casxa"指令[1],软件可以使用这类指令实现锁,通常称这类锁为软件锁。当前一个重要的趋势是在处理器设计中提供硬件实现的锁变量存储单元,并提供精简的锁操作指令,基于这些锁操作指令可以构造高效的锁访问模块,提升多线程并行应用的性能。

本文首先介绍了多核多线程FT处理器硬件锁的特点并与软件锁机制进行比较,然后介绍了用硬件锁构造基本同步模块的方法。其次,基于GNU OpenMP库实现了使用FT处理器硬件锁的同步机制。最后,通过同步较频繁的OpenMP并行应用,对使用硬件锁和使用软件锁进行同步的性能进行了评估和分析。本文的工作对设计、使用硬件锁机制来提升高并发应用的性能具有非常重要的参考意义。

2 背景

2.1 多核多线程FT处理器

共享多级存储层次的片上众核处理器体系结构,是当前及未来相当长时期内通用高性能微处理器的主流体系结构。对于这种共享存储片上众核处理器,多线程并行是最直接、最高效的并行执行模式,甚至其中的处理器核本身就可能在硬件层面上支持多线程。因此,该类处理器是一种高并发度的片上并行系统。

基于锁的同步机制是最广泛使用的同步机制,有大量的遗留软件代码使用这种同步方式。锁是多线程并行程序在用户数据空间中定义的数据元素,通常各个并发的线程利用处理器提供的比较并交换指令完成对锁的原子操作。因此大量线程在高度并发的条件下对锁变量的并发访问开销是高并发并行程序同步开销的主体。

图1是国防科技大学自主实现的16核64线程(每个处理器核以轮转方式支持4个线程)的FT处理器内核及高速缓存结构框图。使用硬件锁提高此类处理器上并发应用的性能,是该处理器的创新点之一。

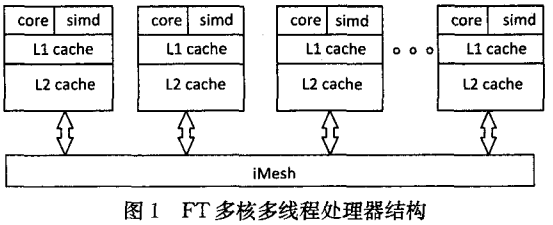

图1 FT多核多线程处理器结构

到稿日期:2012-11-23 返修日期:2013-02-24 本文受国家自然科学基金项目(61170046,61170045),国家863计划项目(2012AA010903)资助。
李春江(1974—),男,博士,副研究员,硕士生导师,主要研究方向为计算机体系结构、编译及优化技术,E-mail:chunjiangli@gmail.com;唐 滔(1989—),男,博士,助理研究员,主要研究方向为编译及优化技术;杨灿群(1968—),男,博士,研究员,硕士生导师,主要研究方向为编译技术、系统软件。

2.2 FT处理器硬件锁的特点

FT处理器的硬件锁位于处理器芯片上，同样的锁访问操作仅通过地址的不同来区分是“读并加一”、“读并减一”或是“读原值”操作。硬件锁地址处于I/O空间，因此所有对硬件锁的读取并不进入各级高速缓存(Cache)；并且硬件锁在芯片内由寄存器文件实现，所以硬件锁用于高并发应用可以获得一定的性能提升。

FT处理器仍然继承使用原有指令集中的比较并交换指令(如引言中所述的“casa”和“casxa”指令)，在使用比较并交换指令时，用户空间的数据变量作为锁变量，锁变量同一般的数据变量一样，要进入各级高速缓存，这在多个处理器核上的不同线程并发访问锁的时候会频繁触发高速缓存作废操作，影响多线程同步的性能。

从Intel 486处理器开始，Intel的X86处理器都具有锁定一个特定内存地址的能力，当这个特定内存地址被锁定后，它就可以阻止其他的系统总线读取或修改这个内存地址。这种能力是通过LOCK指令前缀再加上一些汇编指令来实现的。可以和LOCK指令前缀一起使用的指令包括：BT、BTS、BTR、BTC、XCHG、XADD、ADD，OR，ADC，SBB、AND、SUB、XOR、NOT、NEG、INC、DEC[2]。其中XADD指令能够保证在多处理器系统下的原子操作，它们总会使“LOCK#”信号有效来锁定内存地址，而不管有没有LOCK前缀。那么XADD类指令就可以实现单条指令对指定存储区域的原子操作，其存储区域可以是整个程序的存储空间。但是，FT处理器的硬件锁与此不同，FT处理器的硬件锁地址处于处理器的I/O地址空间，并且数量固定，使用指令集中一般的Load指令实现对锁变量的原子操作，仅根据地址的不同来实现“读并加一”、“读并减一”或“读原值”操作。

2.3 FT处理器硬件锁的特点

FT处理器在IO地址空间中开辟了锁变量存储区域(物理实现上位于处理器芯片内)，对其中锁变量的读取由硬件实现了原子操作功能。

例如图2所示的汇编代码中，对地址为0x8F00000000的64位锁变量，当对“该地址或0x400”后的地址读取时，将锁变量的值读入寄存器%l1的同时存储器中的锁变量自动减一；当对“该地址或0x800”后的地址读取时，将锁变量的值读入寄存器%l1的同时存储器中的锁变量自动加一；当对该地址读取时，仅将该地址内的锁变量读入寄存器%l1，存储器中锁变量的值不变。

```
#define LOCK_ADDR0 0x8F00000000
setx LOCK_ADDR,%l1,%l2
setx LOCK_ADDR|0x400,%l1,%l3
setx LOCK_ADDR|0x800,%l1,%l4
ldx [%l4],%l1 读并加一
ldx [%l3],%l1 读并减一
ldx [%l2],%l1 读出锁变量的内容
```

图2 硬件锁操作的基本特点

栅栏同步和临界区保护是广泛使用的同步机制。栅栏同步就是要所有线程都到达栅栏同步点后，再继续执行后续的程序代码，一旦有线程没有到达，那么先到达的线程就反复判断栅栏同步退出条件，直到条件满足。临界区是一段程序代码，被一个锁变量保护起来，同一时刻只有获得了锁的线程才能执行。临界区通常用0/1锁保护，只有获得了锁的线程才能进入临界区执行。

利用FT处理器提供的硬件锁机制实现的栅栏同步操作模块的汇编代码如图3所示。

```
cmp_thread_sync_inc:
  ldx [%l4], %l1
barrier_wait_sync_inc:
  cmp %l1,64
  bne %xcc, barrier_wait_sync_inc
  ldx [%l2], %l1
```

图3 利用硬件锁实现的栅栏同步

在图3所示代码模块中，通用寄存器%l4中保存的是硬件锁变量所在的存储器位置的“读加一”地址，汇编语句“ldx [%l4],%l1”将该变量的内容读入到通用寄存器%l1中，同时按照硬件锁的实现特点在读取了该锁变量的同时存储器中的该锁变量增加了一。汇编语句“cmp %l1,64”中的64为同步线程数目，表示有64个线程要进行栅栏同步，语句的意思是比较锁变量的值是否和预设的线程数目相同。所有线程都要执行上面的栅栏同步操作模块，当所有线程都执行完这段代码后，所有线程才能继续执行后续的程序语句，先到达的线程要反复读取锁变量的内容，和线程数比较，如果不等就跳转到barrier_wait_sync_inc标号语句反复执行。汇编语句“bne %xcc,barrier_wait_sync_inc”的含义是针对上面做比较的汇编语句“cmp %l1,64”的结果(结果在条件码寄存器%xcc中)，如果不相等就跳转到barrier_wait_sync_inc标号。在Sparc V9指令集中，跳转语句后的语句也在跳转之后立即执行(跳转指令有一个指令空槽)，即上面代码模块中“ldx [%l2], %l1”语句也在不相等的时候执行，该语句中的通用寄存器%l2保存的是硬件锁变量的“读原值”地址。

而同样的功能，如果使用FT处理器的“比较并交换指令(casa或casxa)”，需要的汇编指令代码如图4所示。

```
cmp_thread_sync_inc:
  ldx [%l2], %l1
  add %l1, 1, %l3
  casxa [%l2], %l1, %l3
barrier_wait_sync_inc:
  cmp %l1, 64
  bne %xcc, barrier_wait_sync_inc
  ldx [%l2], %l1
```

图4 用比较并交换指令实现的栅栏同步代码

从图4可见，和使用硬件锁相比，使用比较并交换指令实现的软件锁，需要至少额外执行两条指令。在多线程的环境下，为了避免各个线程频繁读取和比较锁变量，常常要加入一些执行等待操作的指令，那么增加的指令数会远远大于两条。这就是使用FT处理器设计实现硬件锁的优势所在。

图5给出了使用硬件锁实现临界区保护的代码，语句“ldx [%l4],%l5”将通用寄存器%l4中的锁变量地址所指的锁变量内容读入到通用寄存器%l5中，同时存储器中该锁变量值加一。汇编语句“cmp %l5,0”的含义是：由于用硬件锁实现临界区保护，那么锁变量不为0表示已经有线程加锁了，其他线程就反复读取比较，如果锁变量为0，那么进行读取操作，同时已经完成了锁变量增一的操作(即加锁操作)，表示执

行该读取操作的线程获得了锁，可以执行临界区的代码了。获得了锁变量的线程执行完临界区的代码后，在临界区代码的结尾部分，用指令将锁变量重新置为0；那么，其他线程就可以再获得这个锁，进入临界区。

```
require_lock:
  ldx [%l4], %l5
  cmp %l5, 0
  bne require_lock
  nop
```

图5 利用硬件锁实现的临界区保护

3 基于硬件锁实现多线程同步

3.1 多线程同步库

OpenMP并行编程模型和编程接口[3]已经成为共享主存多处理器系统并行编程的事实标准。OpenMP在编程语言级提供指导命令，然后由支持OpenMP的并行编译器将OpenMP并行程序编译成在运行时动态创建多线程并实现多线程并行的程序。

OpenMP编程接口有编程指导语句和运行时库接口两部分，目前产品化的编译器都实现了对OpenMP并行程序的编译支持；并且OpenMP接口规范仍在学术界和产业界的推动下不断发展。

对于OpenMP编程而言，在OpenMP编程接口中定义了最常使用的同步接口“#pragma omp critical”、“#pragma omp barrier”和“#pragma omp taskwait”，而这3个指导语句都会被编译器编译为对OpenMP运行时库中临界区保护同步(critical)和栅栏同步(barrier)的调用。

3.2 使用硬件锁的同步库实现

在用比较并交换原子指令实现锁(即软件锁)的处理器平台上，通常用一个通用的整型数表示多线程互斥访问的锁变量。例如在GNU OpenMP运行库中通常采用如下定义锁变量的方式[4]：

gomp_mutex_t lock __attribute__((aligned(64)))

而gomp_mutex_t由“typedef int gomp_mutex_t”定义。

在FT处理器的硬件锁设计中，在处理器芯片上设计实现了128个64位的锁存储单元(硬件锁变量的物理存储由寄存器文件实现)，并设计实现了相应的硬件锁操作机制，如本文第2节所述。为了在GNU OpenMP运行库中实现使用硬件锁的同步操作，本文做了如下工作。

·硬件锁的管理

FT处理器的硬件锁在IO空间(地址范围0x8F00000000－0x8F00000BFF)中，在芯片上实现了128个64位寄存器，对这一地址范围内整数变量的读取操作实现了如前文所述的硬件锁语义。由于这些硬件锁由全芯片共享，需要由操作系统统一管理。操作系统统一管理硬件锁资源，提供系统调用，实现硬件锁的申请和释放。具体接口如下：

int syscall(355,lock)：申请一个64位锁变量

int syscall(356,lock)：释放一个64位锁变量

其中，lock为指向64位锁变量的地址。

·在GNU OpenMP库中锁变量的申请和释放

基于操作系统提供的硬件锁申请和释放接口，在OpenMP中实现了硬件锁申请和释放的接口：void gomp_hard_malloc_64(unsigned long * lock)和void gomp_hard_free_64(unsigned long lock)。

·使用硬件锁实现栅栏同步

在GNU OpenMP运行库中，栅栏同步结构中定义了多线程进入栅栏同步访问的共享锁变量。栅栏同步的数据结构如图6所示，其中的awaited就是多线程共享访问的锁变量的地址；当运行库使用系统提供的硬件锁变量时，增加了对awaited的定义。

```
typedef struct
{
  unsigned total __attribute__((aligned(64)));
  unsigned generation;
  #if defined USE_HARD_LOCK || defined USE_HARD_LOCK_
  SINGLE
  unsigned long awaited __attribute__((aligned(64)));
  #else
  unsigned awaited __attribute__((aligned(64)));
  #endif
} gomp_barrier_t;
```

图6 GNU OpenMP中栅栏结构

在运行库中，当对栅栏同步初始化的时候，使用gomp_hard_malloc_64(&(bar->awaited))来为此锁变量分配硬件锁。如果硬件锁分配不成功(所有的硬件锁资源已经用完)，则用结构中原来的软件锁变量。

·使用硬件锁实现临界区

在GNU OpenMP运行库中，临界区的保护实现比较简单，在临界区开始处申请一把全局锁(0/1锁)，进入临界区加锁(锁变量设置为1)，退出临界区则解锁(锁变量设置为0)。为了利用处理器的硬件锁机制，调用操作系统的硬件锁分配接口来分配硬件锁。

·使用硬件锁的原子操作的实现

在多线程同步库的gomp_barrier_wait_start函数中，调用了实现“取并加一”的函数。在处理器未支持硬件锁时，“取并加一”是通过调用内嵌函数__sync_add_and_fetch来利用处理器的“比较并交换”指令实现。该内嵌函数是GCC编译器中实现的同步类内嵌函数，关于GCC的内嵌函数请参见文献[5]。

为了利用FT处理器中设计实现的硬件锁，该函数用仅包含一条指令的内嵌汇编来实现：

```
__asm volatile("ldx [%1],%0":"=r"(ret1):"p"((bar
->awaited)|0x400))
```

bar->awaited是栅栏同步中多个变量公用的锁变量的地址，在使用硬件锁的库中，用上述gomp_hard_malloc_64函数来分配。

临界区同步时，在最底层使用硬件锁的指令层面，和栅栏同步完全相同。

4 性能评估和分析

栅栏同步涉及多个线程对锁变量的并发访问，因此本文采用使用栅栏同步的OpenMP并行程序来评估硬件锁的性能。本文分别编译了使用硬件锁和软件锁的GNU OpenMP

(下转第60页)

[9] Kari C, Kim Y A, Russell A. Data Migration in Heterogeneous Storage Systems [C]// International Conference on Distributed Computing Systems. 2011:143-150

[10] Ghemawat S, Gobioff H, Leung S T. The Google File System [C]// The 19th ACM Symposium on Operating Systems Principles. 2003:29-43

[11] Dowdy W, Foster D. Comparative Models of the File Assignment Problem [J]. ACM Computing Surveys, 1982, 14(2):287-313

[12] Serpanos D N, Georgiadis L, Bouloutas T. Mmpacking: A Load and Storage Balancing Algorithm for Distributed Multimedia Servers [J]. IEEE Transactions on Circuits and Systems for Video Thechnology, 1998, 1(8):13-17

[13] Rowstron A, Druschel P. Storage Management and Caching in PAST, a Large-scale, Persistent Peer-To-Peer Storage Utility [C]// The 18th ACM Symposium on Operating Systems Principles. 2001:188-201

[14] Wei Q, Veeravalli B, et al. CDRM: A Cost-effective Dynamic Replication Management Scheme for Cloud Storage Cluster [C]// IEEE International Conference on Cluster Computing. 2010: 188-196

[15] Xie C, Cai B. A Decentralized Storage Cluster with High Reliability and Flexibility [C]// The 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. 2006:116-123

[16] Wang W, Zhao Y. A Novel Network Storage Scheme: Intelligent Network Disk Storage Cluster [C]// IEEE International Conference on Networking, Sensing and Control. 2008:142-147

[17] Kim J, Chou J, Rotem D. Energy Proportionality and Performance in Data Parallel Computing Clusters [C]// International Conference on Scientific and Statistical Database Management, LNCS 6809. 2011:414-431

[18] Thereska E, Donnelly A, Narayanan D. Sierra: Practical Power-proportionality for Data Center Storage [C]// Eurosys. 2011: 169-182

[19] Amur H, Cipar J, et al. Robust and Flexible Power-proportional Storage [C]// The 1st ACM Symposium on Cloud Computing. 2010:217-228

[20] Nakamura S, Shudo K. MyCassandra: a Cloud Storage Supporting both Read Heavy and Write Heavy Workloads [C]// The 5th Annual International Systems and Storage Conference. 2012:14

(上接第 37 页)

库 Libgomp 版本，比较同一 OpenMP 并行应用使用处理器硬件锁和软件锁的性能差异。

实验平台为实现了硬件锁的 FT 处理器仿真器，该处理器有 16 核 64 线程(每核 4 线程)。使用的 OpenMP 并行程序为 NPB[6] 测试集中的 LU 和 MG，以及专门用于测试的 OpenMP 栅栏同步的测试用例 Syncbench[7]。表 1 列出了在不同线程数目、不同同步次数情况下，OpenMP 程序在使用硬件锁和使用软件锁情况下的性能差异。

表 1 使用 OS 管理的硬件锁时应用加速比

程序	并行线程数	Barrier 同步次数	加速比 (硬件锁/软件锁)
Lu. A. x	32	73791	0.8%
Mg. A. x	32	25087	0.2%
Syncbench	32	6057503	1.6%
Lu. A. x	64	147583	1.2%
Mg. A. x	64	50175	0.6%
Syncbench	64	12115007	2.4%

可见，对于科学计算类的应用程序，如 NPB 中的 Lu. A. x 和 Mg. A. x，使用处理器提供的硬件锁，加速效果非常不明显；如果把多次运行的误差考虑在内，使用硬件锁几乎没有加速效果。对于专门用来测试同步的 Syncbench 题目，使用硬件锁，性能稍有提高。

性能提高不明显的一个重要原因是硬件锁的申请和释放需要调用操作系统的系统调用来完成，这会给使用硬件锁带来比较大的系统开销。对于同步数非常多的高并发应用，如果可以考虑在库中固定硬件锁地址(固定使用几个硬件锁)，省去调用操作系统来分配硬件锁和释放硬件锁的过程，可以获得更进一步的性能提升。

表 2 针对 Syncbench 应用，采用使用固定地址的硬件锁方法进行了性能对比测试。可见，使用硬件锁的性能加速比有一些提高。

表 2 使用固定硬件锁的应用加速比

程序	并行线程数	Barrier 同步次数	加速比 (硬件锁/软件锁)
Syncbench	32	6057503	2.2%
Syncbench	64	12115007	3.2%

综合以上测试，可见针对 FT 处理器实现的这种硬件锁机制，应用程序必须在并发度非常大的情况下，才能获得明显的性能提升。

结束语 在多核多线程的高并行度处理器中，针对同步操作提供硬件实现的加速机制是处理器设计中的一个趋势。本文针对 FT 处理器提供的硬件锁机制，在多线程库中使用该处理器的硬件锁实现了栅栏同步机制，并对效果进行了评估和分析。本文的工作对处理器设计以及多线程并行库的设计有重要的参考价值。

参 考 文 献

[1] UltraSPARC Architecture 2007 [S]. Draft D0. 9. 1, 01 Aug 2007:423

[2] Intel 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes: 1, 2A, 2B, 3A and 3B [S]. Intel Corporation, May 2011

[3] OpenMP Application Program Interface. Version 3.0 [S]. OpenMP Group, May 2008. http://www. open mp. org/

[4] The GNU OpenMP Implementation [OL]. http:/gcc. gnu. org

[5] 李春江，杜云飞，易会战，等. GCC 中内嵌函数实现剖析[J]. 计算机科学，2012(6A)

[6] NASA Parallel Benchmark 3. 3. 1[OL]. http://www. nas. nasa. gov/publications/npb. html

[7] Syncbench [OL]. http://trac. mcs. anl. gov/projects/perfor mance/browser/benchmarks/openmpbench-C-v2/syncbench. c? rev =666