第15卷 第8期 2015年3月
1671—1815(2015)08-0099-05

科 学 技 术 与 工 程
Science Technology and Engineering

Vol. 15 No. 8 Mar. 2015
© 2015 Sci. Tech. Engrg.

共享内存并行编程最优同步方法的研究

王 凯 杨剑锋* 郭成城 于银菠
(武汉大学电子信息学院, 武汉 430072)

摘 要 共享内存并行编程是并行编程开发与研究的标准之一,为了保证程序运行的准确性和线程执行的高效率,对共享内存同步机制的研究具有重要意义。互斥锁与事件同步是 OpenMP 标准中的混合同步机制,而斯坦福大学的 Baek 等人扩展 OpenMP 提出的面向事务存储的 OpenTM 应用编程接口,事务内存同步是其突出特点。通过研究发现混合同步存在部分互斥锁同步会严重消耗系统空间资源、线程资源利用率有待提高等问题,而事务同步忽视不可恢复行为的影响。因此,提出一种基于共享内存的最优同步方法,即优化后的混合同步和事务内存同步的结合体,实验论证此优化方法的可行性,程序运行更加高效。

关键词 混合同步 事务内存 同步优化 OpenTM
中图法分类号 TP311; **文献标志码** A

多核架构是实现计算处理速率高速增长的必然趋势,当程序被多个处理器并行执行时,需要必须的同步机制保证程序在数据竞争时能够得出正确的结果,并且在适当的时候控制线程的执行顺序。在共享内存并行编程中,目前使用广泛的同步机制主要是互斥锁与事件组成的混合同步和事务内存两种同步机制。

针对锁和事件同步暴露出的一些缺点,比如主副线程切换、非必要线程等待等浪费大量的系统资源和时间,有学者提出一种基于数据相关性的同步优化算法[1],可以消除基本块当中的冗余同步。面对传统并行程序设计中锁机制的缺点,文献[2]扩展 OpenMP,提出一种面向事务存储的 OpenTM 应用编程接口,避免锁机制可能会导致阻塞式等待和死锁等问题,文献[3]提出面向 OpenTM 应用的并行数据重用理论,体现事务内存保证数据的准确性能优越。文献[4,5]通过研究互斥锁和事务内存的语义和逻辑架构,并算法验证并行程序的软件事务内存同步机制。

基于以上理论研究,首先对混合同步进行优化,调整程序同步方法结构,提高并行程序运行效率。其次,优化事务同步框架,消除不可回退行为造成系统的安全威胁。最后,将对这两种同步机制优化进行分析,并提出共享内存并行编程最优同步方法。

2014年4月28日收到 国家高技术研究发展计划(2012AA010904)、四川省教育厅项目(13ZA0296)资助
第一作者简介:王 凯(1989—),男,湖北黄冈人,硕士。研究方向:并行编程。E-mail:731541109@qq.com。
* 通信作者简介:杨剑锋(1976—),男,副教授。研究方向:并行编程、无线网络。

1 同步机制

1.1 混合同步

OpenMP 标准支持两种不同类型的线程同步,一种是互斥锁,另外一种是事件通知。互斥锁的操作针对需要保护的数据而言,在产生了数据竞争的内存区域加入互斥,包括 critical、atomic 等语句以及 mutex 函数,即对某一块代码操作进行保护,以保证同时只能有一个线程执行该段代码。而事件机制则是通过同步屏障(barrier)、定序区段(ordered sections)和主程序执行(master)等预处理器指示符声明来控制规定线程的执行顺序,调度多线程完成指定任务。

混合同步给并行编程带来极大的方便,拥有锁和事件同步的优点。例如,锁同步方式被编程人员所熟悉,并行编程效率高,而且锁保护的操作类型比事务内存多[4]。事件同步方法语义简明,线程调度明确,逻辑结构清晰。

然而,无论是互斥锁同步还是事件同步,它们都需要开发人员在源代码中加入专用的指导语句,使得多核并行编程复杂度提高。同时,在锁同步进行复杂并行编程的过程中也暴露了许多缺陷,比如当多个线程因相互争夺对方资源,而导致阻塞,可能导致死锁。事件同步一些同步操作会使得空闲线程处于等待状态,导致系统总体运行效率不高。

1.2 事务内存同步

事务内存是数据的集合,程序开发者只需要将事务内存的操作封装为事务,无需考虑复杂的同步问题,事务内存系统负责正确执行事务。OpenTM

是在 OpenMP 标准上,引入事务内存语义,从而提出了一种面向事务存储的应用编程接口[3]。当程序检测到数据不匹配或者程序冲突时,通过 abort( )废除当前事务并由 retry( )回滚到事务开始处重新执行。实现构件 transaction 可以在 OpenMP 并行区内使用[2],语法如图 1 所示。

```
#pragma omp transaction [clause[[,] clause]…]{
    #pragma omp transaction
    Code-in-transaction1;
    If(queue.status==EMPTY){
    omp_watch(addr);
    omp_retry();
    }else{
    t=dequeue(queue);}
    #pragma omp transaction
    …}
```

图 1 OpenTM 事务语法

Fig. 1 OpenTM transaction syntax diagram

与混合同步相比,事务内存同步方法性能更优。首先,事务内存同步具有事务的原子性和隔离性,即一组内存访问操作要么提交要么夭折,而且不同组操作之间没有影响,因此可以避免死锁。其次,事务内存同步能为并发编程提供细粒度的并发,而使用锁同步方法则需要设计更为复杂的细粒度算法[6]。最后,事务内存同步的实现可以是非阻塞的,所以线程在执行时如果发生延迟或者异常终止时,不会导致其他线程的阻塞。

尽管事务内存具有原子性和隔离性,但是事务的概念决定事务内存并不完美,尚存在一些问题有待解决,例如事务中不能支持类似系统调用等行为,一旦执行则可能难以取回已经释放的资源。

2 同步优化

2.1 混合同步优化

针对混合同步机制中可能存在的锁竞争和系统多线程利用率不高等问题,现采取一种将并行区域整合和扩展的方法对程序结构进行改进,主要对锁和事件通知分别进行优化。

首先,锁保护的代码只允许一个线程执行,这导致其他线程处于空闲状态,而且在一些应用比如科学计算中代码可能会被执行千万次,创建、销毁锁都耗费大量的系统资源,因此可以选择使用临界区和事件同步指导语句 master/single 取代锁同步。然而,不是所有的锁都能被替代,临界区虽然比较节约线程上下文切换带来的系统开销,却只适用于同一个进程内。master 语句要求代码只由主线程来进行,single 表示只有一个线程执行一次,因此它们只能在特定的场合取代锁同步[7,8]。

其次,事件同步给并行编程带来极大的便利,各种编译指导语句可以组合使用,如果使用得当可以大大提高程序执行的效率。而且,在每一个并行结构(omp parallel)和工作共享结构(omp for)的结束都隐含着一个 barrier 同步点,要求所有线程必须执行完成调度任务,才能继续向下执行。这种非必要的线程等待既浪费系统多线程资源,又影响程序执行效率,因此采用基于数据相关性的优化方法来消除 barrier 同步。

图 2 是并行编译生成的一个 OpenMP 程序片段,包含三个并行区域,而且区域之间存在关联。然而,并行区域中 d 的循环计算与前一区域无数据相关,故可以将 d 的计算模块合并到第一并行块中。同理可推广将第二、三并行块合并。并行区域合并后,程序结构逻辑更加严密,同步次数减少。

```
#paragma omp parallel
{
#paragma omp for
for(i=0;i<K;i++)
z[i][K+1]=e[i]
#paragma omp for
for(i=0;i<K;i++)
g[i]=e[i]/h;
}
#paragma omp parallel
{
#pragama omp for
for(i=0;i<K;i++)
for(j=i+1;j<K;j++)
d[j]=d[i]+e[i]*a[j][i];
#paragma omp for
for(i=0;i<K;i++)
for(j=i+1;j<K;j++)
t[j]=z[i][j]*g[i]+e[i];
}
#paragma omp parallel for
for(i=0;i<K;i++)
for(j=i+1;j<K;j++){
m[i]=d[j]+g[j];
n[j]=g[i]+a[j][i];
}
```

图 2 并行区域合并

Fig. 2 Parallel region merging

图 3 是并行程序中常见的并行与串行过渡的程序片段。串行区域嵌套在并行区域之间,而且区域之间相互独立,因此扩展并行区,整合成一个大的并行区域,同时采用#pragma omp master 语句替换锁保护操作,调度主线程完成串行计算,并消除冗余的同步,调度其他线程执行其他任务。然而,当串行计算与并行区之间存在数据相关时,相应的 barrier 用于保证所有线程完成任务后才执行后续代码,从而保障数据安全。

```
#paragma omp parallel for
for(i=0;i<K;i++)
Z[i][K+1]=e[i];
//串行计算
omp_set_lock(&mylock);
for(j = 0;j < K;j++){
    t[j]=z[i][j]*g[i]+e[i];
}
omp_unset_lock(&mylock);
#paragma omp parallel for
for(i=0;i<K;i++)
for(j=i+1;j<K;j++)
d[j]=d[i]+e[i]*a[j][i];
```

图 3 整合优化串行区域

Fig. 3 Integration and optimization serial regional

综上所述,程序结构能够调整有两个必要条件:第一,并行区域至少存在两个数据不相关的并行计算代码块;第二,至少存在一个与串行计算数据独立的并行计算代码块。因此,得到并行区域合并和整合的总体框架,如图 4 所示。该结构中存在多个

并行区域和串行区域,采用并行区合并和扩展的方法,将程序整合为一个#pragma omp parallel 并行计算结构,同时对数据相关的计算模块进行同步保护,保证数据的准确性。

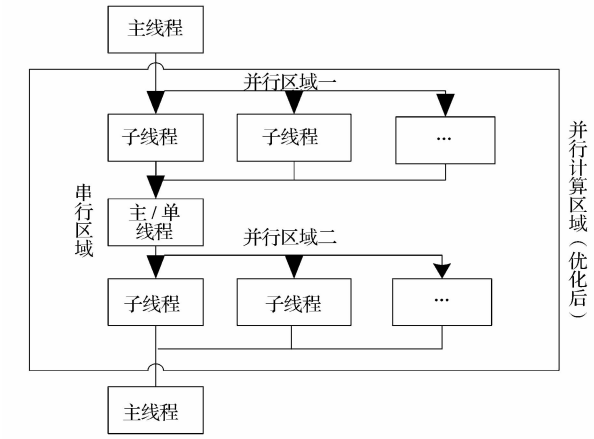

图 4　优化前后结构对比图

Fig. 4　The structure of before and after the optimization

2.2　事务同步优化

事务内存同步机制的优点是事务程序执行过程中如果出现异常,则返回事务最开始处重新执行,直到事务完全正确被执行。然后,当程序中包含一些比如 I/O 操作和释放系统资源等行为时,事务的回退行为无法消除这些行为产生的影响,最终导致同步失败[9]。目前对含有这种不可恢复操作的事务一般有两种处理策略,一是使用缓冲技术[10],二是使所有的事务转为顺序执行[11]。

根据之前对锁的理论分析,锁同步能够保护事务不可恢复的行为,因此考虑对不可恢复的代码块操作采取加锁同步保护。虽然引入的锁机制可能会带来存在死锁的威胁,但是事务内存同步的原子性和隔离性保证程序正确执行,很好地解决了这种潜在问题。其次,一个事务代码可以拆分为多个小的事务,这样虽然使得程序结构更加复杂,但是在某种意义上却可以提高事物的安全性。

基于以上研究,事务结构优化伪代码如图 5 所示,事务程序运行时,线程进行数据计算,当遇到 I/O 端口数据读或者存储以及释放内存等资源的行为时,采取锁同步方法,同时通过冲突检测结果来判定是否需要回滚。检测通过时,线程继续执行代码,并且越过不可回退代码重设事务回滚操作起点。

提出一种保护不可回退行为,并且规划子事务结构的方法,完善事务内存同步机制,可以避免事务同步因不可恢复行为对系统造成的影响,保证程序执行结果的准确性。

```
#pragma omp transaction{
  Retry_1:  WORK_A();
            ……
            omp_set_lock;
            if(!validate){
            omp_unset_lock;
            omp_retry();
            }else{
            Read/Write_ I/O;
            omp_unset_lock;
            }
            Reset omp_retry()_loop;
            WORK_X();
            ……
            }
```

图 5　优化后事务代码结构

Fig. 5　The code structure after optimization

3　最优同步方法及实验论证

OpenTM 并行编程接口规定不允许在事务中使用 OpenMP 同步构造,比如 critical,atomic,mutex 和 barrier 等,同时在 OpenMP 同步构造中使用事务同步会违背强隔离性[2],但此前提是事务之间存在关联联系,即一个事务内处理的是另一事务相关的程序数据,这种情况的事务不符合并行处理要求,应依次执行。因此,考虑多核多线程并行运算的优势,结合混合同步中并行区域合并和扩展的方法和事务特性的优点,设计出基于共享内存的最优同步方法,如图 6 所示。

```
void main() {
  #pragma omp parallel{
    #pragma omp master
        Work_A(args);
    no_wait();
    #paragma omp single
        Work_B(args);
  }
}
void Work_A(args) {
  #pragma omp transaction{
    Work_A();
    omp_set_lock;
    if(!validate){
    omp_unset_lock;
    omp_retry();
    }else{
    Read/Write_ I/O;
    omp_unset_lock;}
    Reset omp_retry()_loop;
    WORK_X(); }
}
Void Work_B(args){
}
```

图 6　OpenTM 新并行编程结构

Fig. 6　The new parallel programming structure of OpenTM

首先,无论是数据并行还是任务并行,都可以将需要处理的程序归于一类事务,即利用#pragma omp transaction 指导语句进行事务的定义与处理,结合 2.2 节中提出的事务内存优化方法完善同步机制,优化回滚操作对系统的影响。其次,结合 2.1 节中提出的并行区域合并和扩展的优化方法,根据事务之间数据关联关系,对事务结构进行并行区域合并和扩展,节省线程调度时间,加快程序的执行效率。

在结构上,此同步方法的保障数据准确的性能比混合同步更优,同时比事务内存同步的线程利用率要高,还解决了事务机制中回滚操作可能存在重大威胁。在功能上,兼具混合同步机制高效的特点和事务同步编程简单、程序出错概率小的特点,是一

种高效又方便的并行编程同步方式。

为了验证此同步优化方法的可行性,本文采用包括多个并行和串行代码块的程序集作为实验的测试程序,部分循环计算代码块之间存在数据关联,程序并行数据包括循环结构和主线程任务。实验平台为双核处理器,主频 2.1 GHz,实验结果如表 1 所示。

表 1 实验结果图

Table 1 Experimental results

程序段	优化前时间/ms	优化后时间/ms
并行区域 1	8 946	5 672
串行区域 1	185	156
串行区域 2	182	
并行区域 2	16 345	14 312
程序集	25 659	20 151

实验结果表明:优化后各个程序段执行速率得到优化,整个程序集的运行效率较优化前有改善。本文讨论并设计了一种结合混合同步和事务内存的优化方法,采用并行区合并与扩展的方法改进程序结构,同时结合事务特性,旨在提高程序的执行效率和完善事务内存机制,实验结果证明程序集运行效率有提高,论证此同步方法的可行性。

4 结 论

研究混合同步和事务内存两种基于共享内存并行编程的同步机制,通过研究发现两种同步方式存在一定的互补性,因此本文在 OpenTM 同步机制中分别对事务结构内、外两方面进行分析与优化,优化后的 OpenTM 同步机制兼容两者的优点,同时避免了两者的缺点。实验结果表明,在并行编程中,优化后的 OpenTM 同步机制对程序的执行效率有改善作用,性能比混合同步等同步机制优越,是目前基于共享内存并行编程的最优同步方法。

参 考 文 献

1 张 平, 赵荣彩, 李清宝. 基于相关性的同步优化算法. 计算机工程, 2006;31(17): 68—70
Zhang Ping, Zhao Rongcai, Li Qingbao. Synchronous optimization algorithm based on correlation. Computer Engineering, 2006; 31 (17):68—70

2 Baek W, Minh C C, Trautmann M, *et al.* The OpenTM transactional application programming interface. Parallel Architecture and Compilation Techniques, 16th International Conference on, IEEE, 2007: 376—387

3 吴俊杰, 杨学军, 刘光辉, 等. 面向 OpenMP 和 OpenTM 应用的并行数据重用理论. 软件学报, 2010; 21(12): 3011—3028
Wu, Junjie, Yang Xuejun, Liu Guanghui, *et al.* Parallel data reuse theories for OpenMP and OpenTM applications. Journal of Software, 2010; 21(12):3011—3028

4 付 明. 低级并行代码中几种同步机制的验证. 合肥:中国科学技术大学, 2009
Fu Ming. Validation of Several synchronization mechanism of low-level parallel code. Hefei: University of Science and Technology of China, 2009

5 李 勇. 基于软件事务内存的并行程序验证. 合肥:中国科学技术大学, 2011
Li Yong. Validation of parallel program based on the software transactional memory. Hefei: University of Science and Technology of China, 2011

6 贾建斌, 黄 春, 赵克佳. 事务存储并行程序编程接口研究. 计算机工程与科学, 2010; 32(11): 136—140
Jia Jianbin, Huang Chun, Zhao Kejia. The research of transaction memory store parallel programming interface . Computer engineering and science, 2010; 32(11): 136—140

7 戴 晨, 陈 鹏, 杨冬蕾, 等. 面向多核的并行编程和优化研究. 计算机应用与软件, 2013; 30(12):198—202,279
Dai Chen, Chen Peng, Yang Donglei, *et al.* Research of parallel programming and optimization for multicore. Computer Applications and Software, 2013; 30(12):198—202,279

8 王 堃. 基于多核的并行程序设计及优化. 南京: 南京大学, 2012
Wang Kun. Design and optimization of parallel programming based on multi-core. Nanjing: Nanjing University, 2012

9 McKenney P E, Michael M M, Triplett J, *et al.* Why the grass may not be greener on the other side: A comparison of locking vs. transactional memory. ACM SIGOPS Operating Systems Review, 2010; 44 (3): 93—101

10 Nakano J, Montesinos P, Gharachorloo K, *et al.* ReViveI/O: efficient handling of I/O in highly-available rollback-recovery servers. High-Performance Computer Architecture. The Twelfth International Symposium on, 2006:200—211

11 Ni Y, Welc A, Adl-Tabatabai A R, *et al.* Design and implementation of transactional constructs for C/C++. ACM Sigplan Notices, ACM, 2008; 43(10): 195—212

(下转第 110 页)

Theoretical Research on Seismic Performance of RC Frame with Reinforced Masonry Wall in the Flexible Connection

JIANG Jun-ming[1], MA Jian-xun[1*], FU Liang[2]
(Department of Civil Engineering, Xi'an Jiao tong University[1], Xi'an 710049, P. R. China;
Shenzhen Design & Research Institute Co., Ltd[1], Ministry of Machinery Industry[2], Shenzhen 518000, P. R. China)

[**Abstract**] The connection type between reinforced masonry wall and the surrounding frame has a significant influence on the seismic performance of RC frames. Three full-scale, one-bay and one-story RC frame specimens were researched under low-cycle reversed loads. The damage behavior, the force-displacement hysteretic curves and skeleton curves, displacement ductility, stiffness degradation, and energy-dissipation capacity of the specimens were analyzed in the flexible connection types, including connecting button connection type, wall-beam connection type, and wall-column connection type. The results show: compared with connecting button connection type and wall-column connection type, the hysteretic curves of wall-beam connection type are more plump; under horizontal earthquake action, wall-beam connection has better energy dissipation capacity.

[**Key words**] RC frame with reinforced masonry wall flexible connection ont-seismic performance finite element analysis

(上接第 102 页)

Research of the Most Optimal Synchronization Method in Shared-memory Parallel Programming

WANG Kai, YANG Jian-feng*, GUO Cheng-cheng, YU Yin-bo
(School of Electronic Information, Wuhan University, Wuhan 430072, P. R. China)

[**Abstract**] Shared-memory parallel programming is one standard of the parallel development and research, in order to ensure the accuracy of program and high efficiency thread of execution, the research of synchronization mechanisms has important significance. Hybrid synchronization composed of mutex lock and events is the synchronization mechanisms in OpenMP standard. And transactional memory is the salient features of the transaction-oriented storage OpenTM application programming interface, expanded OpenMP interface proposed by Beak of Stanford University. There are some problems in these two synchronization mechanisms, for example some mutex synchronization consumes system storage resource serious and the utilization of thread needs to be improved in hybrid synchronization. And transaction synchronization ignore unrecoverable behavior. Therefore, an optimal synchronization method is presented based on shared memory, the combination method of hybrid synchronization and transaction memory mechanisms. The feasibility of this optimization method is demonstrated by experiments, and the program run more efficiently.

[**Key words**] mixed synchronous transactional memory simultaneous optimization OpenTM