基于OpenMP的事务存储同步语义研究

田祖伟[1,2] 李勇帆[1]

(湖南第一师范学院信息技术系 长沙 410205)[1] (国防科学技术大学计算机学院 长沙 410073)[2]

摘 要 多核处理器环境下必须解决多核处理器的并行编程问题,才能够充分发挥多核处理器的性能。事务存储(Transactional Memory)机制提供了一种在多核环境下程序并行执行和同步的方法。已有的工作已将事务存储扩展到了OpenMP,为程序员提供满足事务原子性、一致性和隔离性的共享存储访问。但当前事务存储的语义并不完善,事务间不能交换中间结果,不能实现锁的部分语义。提出并实现了一种基于开放嵌套的事务存储的同步语义,从而解决了事务间不能交换中间结果的问题,增强了扩展事务存储后OpenMP的并行编程能力。

关键词 事务存储,OpenMP,多核处理器,共享存储并行编程

中图法分类号 TP31 **文献标识码** A

Transactional Memory Synchronization Semantics Research Based on OpenMP

TIAN Zu-wei[1,2] LI Yong-fan[1]

(Information Technology Department, Hunan First Normal College, Changsha 410205, China)[1]

(School of Computer, National University of Defence Technology, Changsha 410073, China)[2]

Abstract A good prarallel programming environment is the key to explore effectively Multi-core processor's performance. Transactional Memory provides a way to execute program parallell and synchronization for Multi-core Processors programming, and it has been extended to OpenMP which supports atomic, consistent and isolated shared memory access. However, the semantics of transactional memory is not perfect. Intermediate results can't be exchanged among transactions, and some semantic of locks can't be realized by transactional memory. Based on open nested transactions, this paper presented and realized a kind of transactional memory synchronization semantics, and solved the problem that the exchange of intermediate results among transactions is infeasible, and enhanced the ability of its parallel programing after extending transaction memory to OpenMP.

Keywords Transactional memory, OpenMP, Multi-core processors, Shared memory parallel programming

1 引言

由于提高处理器频率导致线路延迟、功耗和冷却等问题难以解决,人们通过在同一个微处理器中设计多个处理器核来提高微处理器性能,即多核处理器,希望通过多线程并行来提高性能。多核处理器已成为现代微处理器新的发展方向,许多处理器厂商(如Intel和AMD等)都推出了多核产品。但多核并不意味着高性能,多核处理器环境下必须解决多核处理器的并行编程问题,才能够充分发挥多核处理器的性能。长期以来人们习惯于串行编程,而并行程序设计存在许多困难[1]。因此,为程序员提供有力的并行编程手段成为了当前业界和学术界的一个研究热点。

并行程序设计一直相对滞后[1],编写与调试并行程序复杂困难,导致并行软件开发效率低。因此,要充分发挥多核处理器的性能,需要提供有效的并行编程手段,降低并行编程的复杂度,其中一个关键的问题就是要有效协调多线程对共享变量的访问。传统的使用锁、信号量等实现线程同步的方法都是比较低层次的,有很大的局限性,且使用复杂,容易导致死锁等问题。为了简化编程模型和提高并行计算的性能,人们借鉴数据库系统中事务处理的思想,将事务引入并行计算领域,产生了事务存储(transactional memory,TM),为程序员提供一种具有事务特征的并行执行模型和一种满足事务原子性(atomicity)、一致性(consistency)、隔离性(isolation)的共享存储访问机制。国内有研究较为全面地介绍了事务存储[2]。

OpenMP是共享存储程序设计的事实标准,OpenMP API 3.0[3]基于现有的串行语言C/C++/Fortran,采用制导扩展的形式,通过一组编译制导和库函数,提供循环级和任务级的并行,将任务分布到各个线程并发执行,从而达到提高性能的目的。OpenMP由于其影响的广泛性和自身的优势,被移植到了多核处理器,成为多核编程的主要候选语言之一,已经有研究将事务存储扩展到OpenMP[4,5]。

人们提出事务存储的初衷之一是为了能够提供一种替代锁的并行编程机制。但是目前事务存储的语义并不完善,还不能完全替代锁,某些情况下可能导致事务无法完成。进行了事务存储扩展的OpenMP同样存在这个问题。为解决事务间不能交换中间结果的问题,本文提出并实现了一种基于

到稿日期:2008-11-02 本文受湖南省教育厅优秀青年基金项目(08B014),湖南省科技厅科技计划项目(2008GK3134)资助。

田祖伟(1973-),男,副教授,博士研究生,主要研究方向为编译优化;李勇帆(1959-),男,教授,主要研究方向为计算机软件与理论。

开放嵌套的事务存储的同步语义，通过开放嵌套事务将当前事务的中间结果提供给需要读取该共享变量的事务。但是提交时要求参与这个过程的所有事务都同时提交，以保证对共享变量修改的一致性。

2 事务存储简介

事务存储允许程序员进行共享存储并行编程时将线程中的一段代码声明为事务，执行时事务之间对共享存储空间的访问具有原子性、一致性和隔离性。程序员不需要考虑原来锁同步中的一些问题。多个事务对不同的共享变量可以并发地修改，对相同的共享变量可以进行读写操作。如果都是读操作，则可以并行进行。但是，如果其中至少有一个是写的时候，即发生冲突时，相关的每个事务要么成功完成并提交，改变共享变量的值；要么失败，好像事务根本没有执行过一样，这就是原子性的含义。事务失败作废后，要么重新执行事务代码，要么放弃，不再执行。一致性是事务要为线程提供共享数据的某种存储一致性，保证事务内代码见到的共享数据是一致的。隔离性是事务对共享变量修改的中间结果不能为外界所见。对于并行编程而言，事务能够降低程序员编程难度和程序正确性推理的难度。与锁相比，事务存储具有可扩展性，容易使用，可以避免死锁。根据事务存储的实现不同，可以分为基于软件的事务存储模型（STM）、基于硬件的事务存储模型（HTM）以及软硬件混合的事务存储模型（HybridTM）。

事务存储的原型系统为程序员提供的编程接口大体可以分成两类。程序员通过语言机制使用事务，即只需指明事务代码的范围，由编译器完成后端的处理；或者库调用，即不仅要声明事务代码的范围，还要对共享变量的访问调用事务存储支持的接口函数或者硬件指令。图 1 中 x，y 均为共享变量；a 部分就是语言一级的接口，程序员只需要指明事务代码的范围，编译器完成对实际事务存储的调用，编译器需要识别事务中哪些是共享变量，以便使用事务存储完成对共享变量的访问；b 部分需要程序员手工书写对接口函数或者硬件执行的调用。

```
atomic                    Start_trans();
{if(x!=false)             If(trans_read(x)!=false)
  y=true;)                  trans_write(y,true);
   (a)                    End_trans();
                             (b)
```

图 1 事务存储代码

Herlihy[8] 和 Shavi[9] 分别基于硬件和软件首先提出并实现了事务存储。但是由于运行时开销过大等原因，事务存储在接下来一段时间并没有被继续深入研究。随着处理器设计和制造方向的转变，人们在单个处理器中能够获得更多硬件线程，而且拥有核内的高速互联，线程之间的通讯开销变得很小。这一方面要求有效的并行编程机制，另一方面为事务存储的高效实现提供了条件。事务存储渐渐成为计算机体系结构和编译领域的研究热点。尤其是近年来，著名大学、研究所和业界巨头都展开了研究，取得了很大的进展，开发了很多原型系统。而且事务存储还在实际中被应用，如在 Java，C# 等语言中得到了支持，Sun 公司还计划在明年推出包含事务存储实现的多核处理器 Rock 等等。

目前事务存储已经被扩展到了 OpenMP[4,5]，这些工作也提供和丰富了编程接口，能够有效地开发 OpenMP 和事务存储在并行编程方面的功能。由于事务存储语义存在的局限性，进行了事务存储扩展的 OpenMP 仍然存在一些尚未解决的问题，其中包括不支持事务间执行过程中中间结果的交换，导致在实现某些语义时可能导致活锁。

3 基于开放嵌套的事务存储的同步语义机制

事务存储语义不完善，不能完全替代锁。事务存储的研究中发现[6]，如果简单地将原来并行程序中的锁替换为事务，会出现问题，可能导致事务无法完成。

如图 2 所示，基于锁的实现中，两个线程可以通过对共享变量 flagA 和 flagB 的访问实现同步，然后分别完成对 m 和 n 的更新。在基于事务存储的实现中，根据事务存储的隔离性语义，写的数据要等到事务执行成功提交时才为外界所见，对 flagA 和 flagB 写操作要在两个事务完成时才能够为对方所见，而两个事务完成的前提又是要读到 flagA 和 flagB 被赋值为 true。这是一个矛盾，将导致两个事务都不能完成，造成活锁的现象。而且可以看到在这个情况下，两个事务之间读写了相同的共享变量，存在冲突，不可能同时提交。本文采用基于开放嵌套的事务存储的同步语义来解决这个问题。

flagA, flagB = false;

线程1	线程2
acquire(**m->lock**);	acquire(**n->lock**);
flagA = true;	while (!***flagA***) {}
while (!***flagB***) {}	***flagB*** = true;
//update m	//update n
release(**m->lock**);	release(**n->lock**);

基于锁

flagA, flagB = false;

线程1	线程2
begin_transaction();	begin_transaction();
flagA = true;	while (!***flagA***) {}
while (!***flagB***) {}	***flagB*** = true;
//update m	//update n
end_transaction();	end_transaction();

基于事务存储

图 2 事务存储语义缺陷

嵌套事务是指该事务完全包含于某个事务的动态区域，即一个事务内包含或者通过调用关系间接包含另外一个事务。开放嵌套（open nested）事务的语义是：内层事务和外层事务不会发生冲突，按程序串行执行语义。内层事务与其他事务（非嵌套事务）发生冲突时作废。不会导致外层事务作废，内层事务成功完成并提交后，修改结果可以为其他事务所见。但是当外层事务执行失败作废时，内层事务的执行结果要被撤销。开放嵌套事务能够增加程序的并行性[7]，提高性能。为了保持一致性，保证内层事务的结果不为其他事务使用，有些研究采用了锁的机制[7]。封闭嵌套（close nested）事务与前者的区别是，内层事务提交后，不为其他事务所见，其他情况相同。封闭嵌套事务的优势在于访问冲突激烈的变量时，如果发生冲突，可以减少事务作废的程度，即只作废内层的嵌套事务部分，外层的事务可以保持其进展情况，提高效率。

开放嵌套事务提交后，其结果可以为其他需要读取该结果的事务所见，但是其他事务需要以显式的方式来读取该结果，同时从这个时刻开始外层事务重新侦测该变量的冲突情况。如果其他事务不以显式的方式来读取，那么可能会读取没有被修改的旧值，也可能会检测到冲突而导致某个事务作废重新执行，这与具体实现相关。因为嵌套事务尤其是多层嵌套的情况非常复杂，暂时将嵌套的深度限制为两层，即只允许外层事务中嵌套一层开放或者封闭的嵌套事务。下面我们对开放嵌套的语义进行改进，基于 OpenMP 的事务存储扩展来解决前面提到的问题。

4 基于开放嵌套的事务存储的同步语义机制的实现

OpenMP API 3.0 提出了 task 的概念和关键字，task 适

合于描述不规则的并行[3]，将 OpenMP 对并行的描述能力由主要在循环级扩展到了循环级和任务级，这使得 OpenMP 能够适应更广泛的并行编程应用。已经有研究将事务存储扩展到了 OpenMP[4,5]。我们将本文提出的新语义扩展到 OpenMP API 3.0，设计并实现了相应的接口函数，下面是具体的扩展情况。

pragma omp transaction [clause[[,] clause]…]
structured-block

其中 clause 是 ordered，(open | closed) nested。ordered 要求执行事务的各线程以串行的顺序提交，如果没指明，默认进行未排序的串行事务。open nested 指明该事务是开放嵌套方式，close nested 指明该事务是封闭嵌套方式，默认情况下是 close nested。

为实现相应的语义，我们定义了读取开放嵌套事务提交结果的函数 void read_opennested_tran (varlist)。这个函数的输入是一个共享变量列表，结果分几种情况：如果没有开放嵌套事务更新了这些变量，那么保持当前事务最近见到的这些变量的值。即如果是第一次访问该变量，则从内存读取该变量。如果前面已经读取过了，那么就返回当前事务中该变量的值。如果有开放嵌套事务更新了这些变量或者部分变量，则更新对应变量的值，并且确定当前事务所在的最外层事务，与修改变量的开放嵌套事务所在的最外层事务一起提交。

OpenMP API 3.0 其他主要指导命令的扩展如图 3 所示，为了避免过于复杂的情况，我们不对这些指导命令进行事务的嵌套，这些指导命令一般作为外层事务。transfor 重用 for 的部分从句，某些从句作为扩展或者增加来描述相关循环体的事务特性，基本上对应于原来的含义，可以在其中使用事务指导命令。Section 和 task 指令命令的扩展与 for 类似。

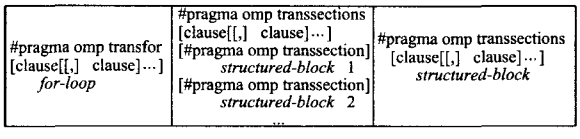

图 3 OpenMP 指导命令的事务存储扩展

现在考虑图 2 中的情况，以基于 OpenMP 的事务存储扩展来解决。如图 4 所示，线程 1 通过 task 指导命令嵌套了一个开放嵌套的事务，将 flagA 的值改为 true。线程 2 通过相应函数读取到了该值，并保证与线程 1 的事务同时推荐，以保持对共享变量修改的一致性，并从此刻开始重新计算对 flagA 变量的冲突检测，因此不会导致外层事务的冲突。线程 2 通过开放嵌套事务将 flagB 的值改为 true，同样线程 1 通过特殊函数读取该变量的值。两个外层事务完成了同步，分别完成了对 m 和 n 的更新。如果其中有外层一个事务与其他事务发生冲突而要作废，那么这两个外层事务都要作废。只有两个外层事务都能够成功提交时，它们的修改才生效，保持对共享变量修改的一致性。

```
flagA, flagB = false;
线程1                                   线程2
#pragma omp transtask                   #pragma omp transtask
 {                                       {
 #pragma omp transaction opennested      read_opennested_tran(flagA);
  {flagA = true;}                         while (!flagA)
 read_opennested_tran(flagB);             {read_opennested_tran(flagA);}
 while (!flagB)                          #pragma omp transaction opennested
  {read_opennested_tran(flagB);}          {flagB = true;}
 //update m                              //update n
 }                                       }
```

图 4 基于开放嵌套事务存储语义的事务同步

结束语 事务存储为程序员提供了一种高层的具有事务特征的并行执行模型和一种满足事务原子性(atomicity)、一致性(consistency)、隔离性(isolation)的共享存储访问机制。但目前事务存储的语义并不完善，事务间不能交换中间结果，不能实现锁的部分语义，某些情况下甚至将导致活锁。本文提出并实现了一种基于开放嵌套的事务存储的同步语义，并且将其扩展到了 OpenMP API 3.0，为事务间同步提供了一种解决方法，增强了事务存储的语义，解决了事务间不能交换中间结果的问题，增强了扩展事务存储后 OpenMP 的并行编程能力。在下一步的研究工作中，我们将进一步探讨在复杂的嵌套情况下，如何更有效地进行事务间的同步问题。

参 考 文 献

[1] 安虹，陈国良. 并行程序设计模型和语言[J]. 软件学报，2002，13(1)：118-124

[2] 彭林，张小强，谢伦国. 事务存储研究[C]// 中国计算机大会论文集. 2008

[3] Meadows L. OpenMP 3.0-A Preview of the Upcoming Standard [J]. Lecture Notes in Computer Science，2007，4782：4

[4] Baek W，Min C C，Trautmann M，et al. The OpenTM Transactional Application Programming Interface[C]// Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques. 2007：376-387

[5] Milovanovic M，et al. Transactional Memory and OpenMP[C]// Practical Programming Model for the Multi-Core Era，Proceedings. 2008，4935：37-53

[6] Blundell C，et al. Deconstructing Transactional Semantics：The Subtleties of Atomicity[C]// Workshop on Duplicating，Deconstructing，and Debunking (WDDD). 2005

[7] Ni Y，Menon V，Adl-Tabatabai A-R，et al. Open nesting in software transactional memory[J]// Proceedings of the 12th ACM SIGPLAN Symposium on PPOPP. 2007：68-78

[8] Herlihy M，Eliot J，Moss B. Transactional Memory：Architectural Supportor Lock-free Data Structures[C]// Proceedings of the 20th Annual International Symposium on Computer Architecture. 1993：289-300

[9] Shavit N，Touitou D. Software transactional memory[J]. Distributed Computing，1997，10(2)：99-116

(上接第 165 页)

[6] Neven F. Automata Theory for XML Researchers [J]. ACM SIGMOD Record，2002，31(3)：39-46

[7] Murata M，Lee Dongwon，Mani M，et al. Taxonomy of XML schema languages using formal language theory [J]. ACM Transactions of Internet Technology，2005，5(4)：660-704

[8] 高军，杨东青，唐世渭，等. 一种基于 DTD 的 XPath 优化方法[J]. 软件学报，2004，15(12)：1860-1868

[9] 高军，杨东青，唐世渭，等. 基于树自动机的 XPath 在 XML 数据流上的高效执行[J]. 软件学报，2005，16(2)：223-232

[10] Bruggemann-Klein A，Wood D. One-unambiguous regular languages[J]. Information and Computation，1998，142(2)：182-206

[11] Hopcroft J E，Motwani R，Ullman J D. Introduction to Automata Theory，Languages，and Computation[M]. Beijing：Tsinghua University Press，2002