# 一种硬件事务存储系统中的事务嵌套处理方案

刘 轶,吴名瑜,王永会,钱德沛
(北京航空航天大学计算机学院中德联合软件研究所,北京 100191)

**摘 要:** 事务存储是一种旨在改善多核处理器可编程性的技术,高效的事务嵌套处理是硬件事务存储系统面临的挑战性问题之一.为了在不显著增加硬件复杂性的前提下实现高效的事务嵌套处理,提出了一种支持有条件部分回滚的事务嵌套处理方案 CPR,该方案为各层嵌套事务维护统一的数据集合,通过增加少量硬件状态位以记录各层嵌套事务对数据的读/写状态,实现了满足特定条件时事务进行部分回滚,在不显著增加硬件复杂性的前提下,提高了闭合嵌套模型的处理效率.基于模拟器 Simics 与多核扩展包 GEMS 进行了模拟实验和性能评价,结果表明,CPR 方案在复杂性显著低于 nested LogTM 的情况下,获得了与其基本一致的性能,同时性能相对于传统的扁平模型有显著提升.

**关键词:** 事务存储; 多核处理器; 可编程性; 事务嵌套; 部分回滚
**中图分类号:** TP302 **文献标识码:** A **文章编号:** 0372-2112 (2014)01-0130-07
**电子学报 URL:** http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2014.01.021

# Supporting Transaction Nesting in Hardware Transactional Memory

LIU Yi, WU Ming-yu, WANG Yong-hui, QIAN De-pei
(*Sino-German Joint Software Institute, School of Computer, Beihang University, Beijing* 100191, *China*)

**Abstract:** Transactional memory is an attractive technology to improve programmability of multi-core processors. However, there still exist challenges for hardware transactional memory including efficient transaction nesting. To support closed nesting efficiently without increasing hardware complexities significantly, this paper proposes a CPR scheme which supports conditional partial rollback on conflict. In stead of rolling back to the outermost transaction as in commonly-used flattening model, the CPR scheme just rolls back to the conflicted transaction itself or one of its outer-level transactions if given condition is satisfied. By adding a series of hardware bits in transactional buffer to record read/write status of each nested transaction, the CPR scheme only maintains a global data set for all of the nested transactions rather than independent data set for each nested transaction as in nested LogTM. Evaluation results show that the CPR scheme achieves similar performance with the nested LogTM, and is better than the flattening model.

**Key words:** transactional memory; chip multiprocessor; programmability; transaction nesting; partial rollback

## 1 引言

随着多核处理器的快速发展,程序需要更多地采用多线程机制以充分利用多核资源来提升程序性能.传统的并发编程模式通过锁、信号量等机制实现进/线程间的同步和互斥,程序员需要花费较多精力考虑同步和互斥问题,还常常由于同步或互斥使用不当导致程序性能下降和死锁,这使得面向多核的并发程序的编写难度和调试复杂性相对于串行程序都要高得多.事务存储(Transactional Memory,TM)是一种旨在改善并行系统可编程性的技术,随着多核处理器的发展,该技术在近年来已成为计算机系统结构领域的研究热点之一.

事务嵌套是硬件事务存储系统面临的挑战性问题之一,本文针对该问题提出了一种支持有条件部分回滚(Conditional Partial Rollback,CPR)的硬件事务嵌套处理方案,该方案基于多核处理器结构和已有的 cache 一致性机制,只需在硬件事务存储中增加少量硬件,便可支持嵌套事务的部分回滚.基于该方案的事务嵌套处理方法在 Simics 模拟器及 GEMS 多核扩展包中实现,并通过 STAMP 测试集及 GEMS 中的测试程序验证其性能.实验结果表明,该方案可有效提高事务嵌套的处理效率.

## 2 事务存储技术及相关工作

事务存储首先在文献[1]中提出,它将程序中的有限机器指令序列作为一个事务(transaction),在系统结构层面保证事务执行的原子性(atomicity),并提供提交

收稿日期:2012-11-02;修回日期:2013-02-28;责任编辑:孙瑶
基金项目:国家自然科学基金(No.61133004,No.61073011);国家 863 高技术研究发展计划(No.2012AA010904)

(commit)、放弃(abort)和回滚(rollback)等操作原语.与传统的并发编程模型相比,TM 技术的优点包括:程序员不必过多地考虑线程间的同步/互斥操作,而只需在程序中进行事务的划分,因而并发程序的可编程性大大改善;事务间的互斥由系统自动完成,必要时可进行回滚,不会出现线程之间由于互斥不当导致死锁的情形;多个事务可以并发地投机(speculative)运行,仅在冲突发生时才进行回滚操作,不会出现一个线程长时间独占资源导致其他线程阻塞的情况,因而程序运行时性能得到了提升.

根据实现方法的不同,TM 可以被分为硬件 TM (Hardware Transactional Memory, HTM)[2~7] 和软件 TM (Software Transactional Memory, STM)[8~12].硬件 TM 系统通过硬件来支持事务的原子性,性能较高且编程方便,但需对处理器及存储系统进行改动,实现较为复杂,且受到硬件资源的限制,如要求事务不能过大(一般要在一个调度时间片内结束).典型的硬件 TM 系统有:TCC[2]、UTM/LTM[3]、LogTM[4]、ASF[5]、DynTM[6]等.软件 TM 系统通过软件来支持事务的原子性,可以基于现有硬件平台实现,并且灵活性好、功能强大,可以对事务提供更灵活的支持,如事务嵌套、部分回滚操作等,但由于事务管理和数据一致性维护的开销较大,性能相对较差.除硬件 TM 和软件 TM 之外,还出现了一些硬软件混合系统结构[13~16].

事务嵌套处理是硬件 TM 面临的挑战性问题之一[17,18].目前多数硬件 TM 系统或者不支持事务嵌套,或者使用效率较低的扁平模型(flattening model)支持闭合式嵌套.已有的提高事务嵌套处理效率的方案是 Nested LogTM[19],该方案同时支持开放和闭合式嵌套,对闭合式嵌套,它回滚到冲突事务而不是最外层事务的开始(即部分回滚),但该方案需要为每一嵌套层维护独立的数据集合,这将大大增加硬件复杂性.与 Nested LogTM 相比,本论文提出的方案只为所有嵌套层维护一个数据集合,因而只需增加少量硬件,付出的代价是必须在满足一定条件时才能进行部分回滚,即支持有条件的部分回滚.

## 3　事务嵌套处理方案

### 3.1　两种事务嵌套模型

事务嵌套是指在一个事务内部存在一个或多个其他事务,这通常出现在事务内部进行子程序调用等场合.目前事务嵌套语义模型主要有以下两种:

(1)闭合式嵌套(Closed nesting)

在该模型中,一个事务和它内部嵌套的所有事务被视为一个整体以保证其原子性,即要么该事务及其所有嵌套事务全部提交,要么全部放弃回滚.硬件 TM 系统中常用的处理方法是扁平模型(flattening model),即事务及其内层嵌套事务被当作一个事务处理,当最外层事务结束时才执行提交操作,当嵌套事务发生冲突时,回滚至最外层事务的开始.这无疑将对系统性能产生较大影响,尤其是当事务嵌套层次较多时,回滚开销较大,同时冲突概率也较大.

(2)开放式嵌套(Open nesting)

在这种事务嵌套模型中,一个事务和它内部嵌套的事务各自分别提交,回滚时也仅回滚到冲突事务的开始.与闭合式模型相比,该模型的效率较高,但对编程人员不透明,因为需要考虑当内层事务成功提交但外层事务因冲突回滚的情形,通常需要编程人员编写额外的处理程序,这显然是与 TM 技术改善并行系统可编程性的初衷相违背的.

### 3.2　支持有条件部分回滚的处理方案

提高闭合式嵌套处理性能的方法是,当事务因冲突回滚时,只回滚到冲突事务的开始而不是最外层事务的开始.Nested LogTM 采用的即是此种方案,这需要为各层嵌套事务维护独立的数据集合,将大大增加硬件 TM 系统的复杂度和实现成本.

为了在不大幅增加硬件复杂度的前提下有效提升闭合式嵌套处理的性能,可以考虑进行有条件的部分回滚(Conditional Partial Rollback, CPR).其思路是:不为各层嵌套事务维护独立的数据集合,而是只维护一个全局数据集合,但在该集合中记录各事务对数据的访问情况.当一个事务冲突回滚时,如果其访问的数据集与外层事务的数据集没有重叠,则可以只回滚到冲突事务的开始;如果与外层事务的数据集有重叠,则回滚到有重叠的最外层事务的开始.

形式化描述如下:设事务嵌套层次为 $n$,最外层事务为 $T_0$,最内层事务为 $T_{n-1}$,各层嵌套事务的写数据集合分别为 $D_0, D_1, \cdots, D_{n-1}$.

当 $c$ 层事务 $T_c$ 发生冲突时,需回滚至 $m$ 层事务 $T_m$,有:

$$m = \mathrm{Min}\{\, i \mid \forall\, i \forall\, j[(D_c \cap D_i \neq \varnothing) \wedge (D_i \cap D_j = \varnothing)]\}$$
$$(i = 0,1,2,\cdots,c;\ j = 0,1,2,\cdots,i-1) \qquad (1)$$

根据式(1),系统将回滚至与事务 $T_c$ 写数据集有重叠的最外层事务,如果 $T_c$ 与其外层事务无数据重叠,则回滚至 $T_c$ 的开始.

图 1 给出了一种有条件部分回滚的例子.根据扁平嵌套模型,当内层事务发生冲突时,将回滚至最外层事务;在 CPR 方案中,如果内层事务与外层事务访问的数据集合无重叠,则只回滚至该事务起始位置,如有重叠则回滚至与其数据集合有重叠的外层事务起始位置.

由以上介绍可以看出,与传统的扁平模型相比,

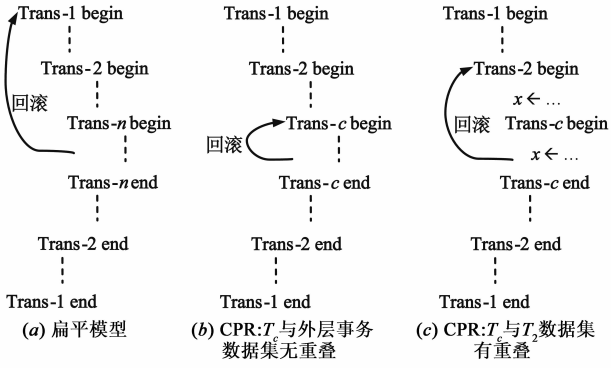



图1 有条件部分回滚示例

CPR 方案通过实现有条件的部分回滚,提高了事务嵌套处理的性能;与另一种事务嵌套处理方案 Nested LogTM 相比,CPR 方案不需要为每个事务嵌套层设置独立的数据集,因而大大降低了硬件复杂度,代价则是只能在满足一定条件时才可以进行部分回滚,需要回滚的层数与各层嵌套事务之间是否共享数据密切相关,当各层嵌套事务共享数据较少时,CPR 方案部分回滚的概率较大,而当各层嵌套事务共享数据较为频繁时,CPR 方案将更多地回滚到更外层事务.

### 3.3 事务支持硬件

支持有条件部分回滚的事务缓冲区结构如图 2 所示.它主要用于缓冲事务执行过程中访问的数据,并记录相关状态,以便回滚或最终提交.事务缓冲区结构与 cache 相似,数据也以行(line)为单位存储,主要区别在于,缓冲数据包括新(new)、旧(old)两个拷贝,其中旧拷贝是指事务启动前的数据,新拷贝则为事务执行过程中的修改版本.每行数据还配有 $n$ 位的读状态向量和 $n$ 位的写状态向量,用于记录该行数据在事务执行过程中的读/写情况.读/写状态向量中的各位分别对应各个不同层次嵌套事务读/写数据的情况.

事务缓冲区与 L1 cache 处于同一层次,两者并列,当处理器执行事务时(即事务状态下)使用事务缓冲区,而在非事务状态下使用 L1 cache.事务状态下,事务缓冲区取代 L1 cache,原有的 cache 一致性机制不做修改,但此时数据写操作只局限于事务缓冲区.直到事务提交时,事务缓冲区中被更新过的行的状态位才被逐行清除,清除操作使得新数据对 cache 一致性机构变为可见.

事务支持硬件中设置一个事务嵌套寄存器(Transaction Nesting Register,TNR),用于记录事务嵌套层次,当处理器开始执行最外层事务时,该寄存器最低位置 1,之后每进入一个嵌套层次,寄存器左移 1 位,每当一个事务结束,寄存器右移 1 位,事务执行过程中,用该寄存器的位映像给事务缓冲区中的读/写向量置位.当事务结束且寄存器最低位为 1,即最外层事务结束时,执行提交(commit)操作.当事务嵌套层次超过 $n$ 时,更深的事务嵌套处理回退为扁平模型.

CPR 方案可以方便地在该硬件结构中实现.当事务需要回滚时,根据事务嵌套寄存器和事务缓冲区中的写向量可以得到该层事务及其他层嵌套事务写过的数据行,进而得到各层事务的写数据集.


图2 支持有条件部分回滚的事务硬件结构

由以上硬件结构可知,CPR 方案需为每行数据增加 $2\times(n-1)$bit 的读/写状态位.按 $n=16$,事务缓冲区容量 16KB、每行 64 字节计算,共 256 行数据,需额外增加 $256\times2\times(16-1)=960$ 字节.与此相比,如为每层嵌套事务维护独立数据集,需额外增加 256 行 × 15 层 × 64 字节 = 240KB.依此计算,CPR 方案所需的额外硬件仅为后者的 0.39%.

## 4 事务存储系统结构

### 4.1 系统结构

基于前述事务支持硬件的 TM 系统结构如图 3 所示.该系统基于多核处理器结构,通过在处理器核内增加部分支持事务执行的硬件,实现程序中事务的高效执行.其中虚线框内是专为事务存储增加的硬件部件,其他部分与传统处理器基本一致.

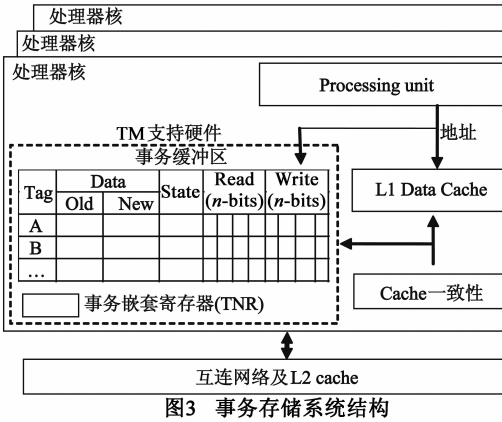



图3 事务存储系统结构

### 4.2 事务执行过程

(1)冲突检测与处理

分属不同并发线程的事务在执行过程中可能因访问共享数据产生冲突,这包括读-写冲突和写-写冲突.

两种情形是:一个线程(事务)读或写了一行数据,随后另一线程(事务)试图写该行数据;一个线程(事务)写了一行数据,其后另一线程(事务)试图读该行数据.

冲突检测基于已有的 cache 一致性机制,如基于总线监听或基于目录的 cache 一致性机制.通过用硬件记录每个行是否被投机性读/写过,就可以知道是否发生了冲突.

检测到冲突后需进行冲突处理,原则是:对发生在事务和非事务代码之间的冲突,事务放弃并回滚;对发生在两个事务之间的冲突,当前请求数据的事务继续,其他冲突事务放弃并回滚.

(2)放弃与回滚

执行回滚操作时,首先根据当前事务嵌套层次和写向量确定需回滚到哪一层次,并将事务缓冲区中该层事务及其嵌套事务写过的行(Write 标志置位)置为无效,仅是投机性读过的行(仅 Read 标志置位)状态不做变动,随后产生一个异常(exception),处理器自动启动一段异常处理程序(exception handler)进行处理,该程序将清除相应标志位,并重新启动事务.

(3)提交

事务执行完毕将进行提交操作,该操作逐行清除事务缓冲区中的读/写标志位,使数据对 cache 一致性机构可见.

### 4.3　指令扩展与编程接口

作为一种硬件 TM 系统,事务处理的多数工作对程序员是透明的,并且没有编程语言种类的限制.系统仅扩展了 2 条机器指令(见表 1),分别对应 2 个接口函数供程序员使用.程序员只需在系统中进行事务的划分,并在事务的开始、结尾处用接口函数进行标示.

**表 1　扩展的事务指令及编程接口**

| 指令 | 描　述 | 接口函数 |
|---|---|---|
| XB | 事务启动 | BEGIN_TRANSACTION() |
| XC | 事务提交 | COMMIT_TRANSACTION() |

## 5　评价与分析

### 5.1　实验环境与实验方法

CPR 方案的实验和性能评价基于系统结构模拟器 Virtutech Simics[20] 和多核扩展包 GEMS[21] 进行.模拟平台采用执行驱动(execution-driven)模拟方式,支持操作系统和应用软件的模拟运行.

目标系统基于 SPARC 处理器结构,并扩展了事务存储相关的硬件部件及指令,处理器核个数为 2 ~ 16 个,目标机运行 Solaris 操作系统,具体配置如表 2 所示.

**表 2　目标机配置**

| 参数 | 配置 |
|---|---|
| 处理器 | Ultrasparc-iii-plus,1GHz |
| Cache 容量 | L1:32KB;L2:4MB;事务缓冲:16KB |
| Cache 行 | 64 bytes |
| 内存容量 | 2GB |
| Cache 一致性协议 | MESI_CMP_filter_directory |
| 互连网络 | Hierarchical switch topology |

测试程序(见表 3)分为两类:第一类共四个程序 genome、kmeans、vacation、yada 来自事务存储系统测试集 STAMP[22],根据配置参数不同,kmeans 分为冲突概率较小的 kmeans-low 和冲突概率较大的 kmeans-high,同理,vacation 也分为 vacation-low 和 vacation-high;第二类为 GEMS 中的双向队列操作测试程序 deque,将其修改为多层嵌套事务,并根据各层嵌套事务间数据共享度大小,分为 share-h、share-m 和 share-n 三个程序,分别代表共享度高、中、低三种情况,同时根据嵌套层数不同,分为 nest1-nest4 四个程序,分别表示嵌套层数为 1 ~ 4 的情况.

**表 3　测试程序**

| 测试程序 | 描　述 | 事务长度 |
|---|---|---|
| genome | 生物信息,基因测序 | 中等 |
| kmeans-low<br>kmeans-high | 数据挖掘,K-means 聚类 | 短 |
| vacation-low<br>vacation-high | 仿真旅行订票在线事务处理(OTP) | 中等 |
| yada | Delaunay 网格细化 | 长 |
| share-h<br>share-m<br>share-n<br>nest1-nest4 | 共享队列上的入队/出队操作 | 短 |

### 5.2　实验结果及分析

图 4 给出了 CPR 方案相对于传统锁机制程序的加速比,可以看出,随着处理器核数的增加,事务程序通过事务的投机运行可以更充分地利用多核资源,从而获得比传统锁机制程序更好的性能.当增加到 16 核时,加速比最高可达 21,远远高于锁机制执行下的效率.由于 yada 程序的事务较长、读写集较大,容易导致溢出,

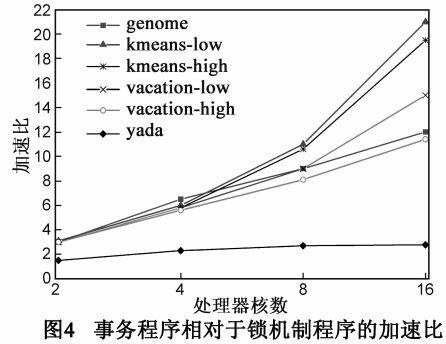



**图4　事务程序相对于锁机制程序的加速比**

同时也会使事务投机执行的成功率受到限制,因而加速比提升较小,在 16 核处获得最大值 2.77.

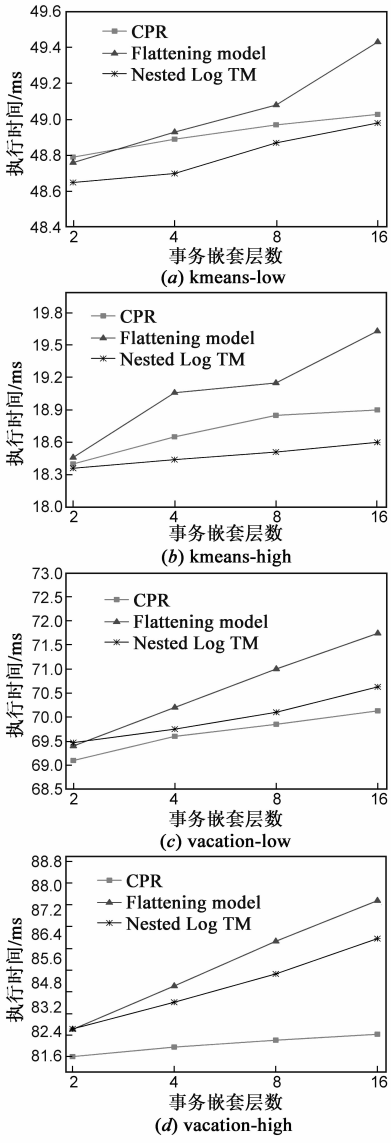



**图5 不同嵌套层数事务的程序执行时间**

图 5 给出了 16 核下几个程序在不同嵌套层次下的执行时间.如图 5 所示,当嵌套层数为 2 时,三种机制下事务冲突后的回滚方式没有区别,都会回滚到最外层事务重新执行,因此程序执行时间相差不大.随着嵌套层次的增加,三种机制的程序执行时间都有所增加,CPR 方案的优势开始明显.vacation 程序中含有一定的长事务,由于 Nested LogTM 采用的是积极版本管理,长事务的回滚容易造成明显的性能下降,对于这种情况,采用懒惰版本管理的 CPR 方案则更为高效,另外,为获得相同数量级的执行时间,两个 kmeans 程序使用了不同的数据文件进行测试.

图 6 给出了嵌套层数及嵌套层间数据共享度对性能的影响.如图 6(*a*)所示,当嵌套层数为 1 时,加速比接近于 1,此时 CPR 方案等同于扁平模型.随着嵌套层次及处理器核数的增加,加速比逐渐增大.当核数达到 16,嵌套层次为 4 时,加速比达到 2.8.原因在于,处理器核数越多,并行执行的线程数增加,冲突概率增大,回滚就更加频繁.同等核数下,嵌套层次越多,部分回滚与回滚到最外层的差别就越明显.如图 6(*b*)所示,CPR 的性能与层间数据共享度有着紧密的联系.如果嵌套层间数据共享度高,部分回滚的概率就低,那么 CPR 方案与扁平模型的加速比就接近于 1,反之,嵌套层间数据共享度越低,部分回滚发生的概率越大,加速比就越大.

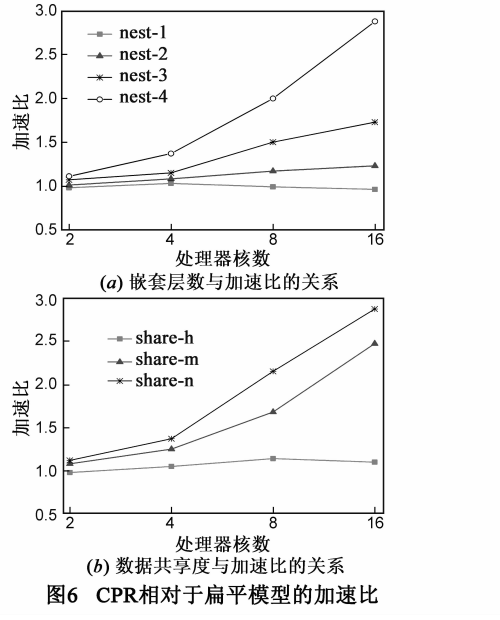
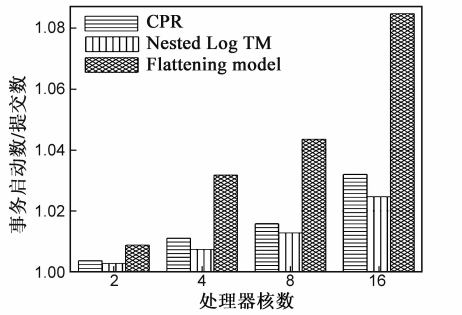
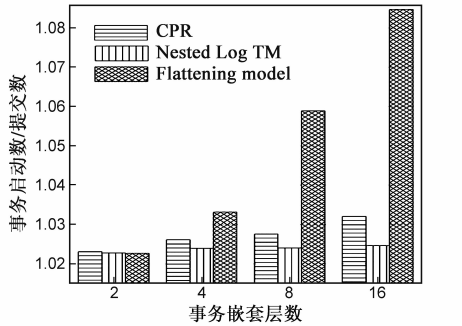



**图6 CPR相对于扁平模型的加速比**

图 7 给出了使用 kmeans-high 程序(事务数:10936)测得的启动事务数与提交事务数之比.图 7(*a*)表示在不同处理器核数下 16 层嵌套事务的启动/提交数,图 7(*b*)表示在 16 核下不同嵌套层数的启动/提交数.当嵌套事务发生冲突时,需要进行回滚并重新启动事务,因此,该值越接近于 1,表明事务回滚次数越少,回滚影响范围越小.如图 7(*a*)所示,随着处理器核数的增加,扁平模型的启动/提交次数比迅速增加,而 CPR 和 Nested LogTM 则增长得较为缓慢,这是因为随核数增加,事务并行度增大,冲突发生更加频繁,而部分回滚可以有效缩小回滚范围.由图 7(*b*)可知,随着嵌套层次增加,扁平模型回滚次数明显增加,而 CPR 和 Nested LogTM 采用的部分回滚方式可大大减小回滚代价,因而比值维持在较低水平.从图 7 还可以看出,由于 CPR 方案需要满足一定条件时才能进行部分回滚,其比值略高于 Nested LogTM.




(*a*) 不同处理器核数下16层嵌套事务的启动/提交数




(*b*) 不同嵌套层数下事务的启动/提交数

**图7　事务启动/提交数**

## 6　小结

事务存储能够改善多核处理器可编程性,避免死锁,并提高并行程序的性能.但它也面临着诸如事务缓冲区溢出、事务嵌套处理、事务内 I/O 操作等一系列挑战性问题.

针对硬件事务存储系统中事务嵌套处理问题,本文提出了一种支持有条件部分回滚的 CPR 方案,该方案基于多核处理器结构和现有的 cache 一致性机制,可以在不显著提高硬件复杂度的情况下,对事务嵌套进行灵活处理.与目前硬件事务存储系统中普遍采用的扁平嵌套方式相比,CPR 方案可以在满足特定条件时进行部分回滚,有效地提高了性能.此外,它只为所有嵌套层维护一个数据集合,比 Nested LogTM 的硬件复杂度显著降低.模拟实验数据表明,CPR 方案相对于传统的扁平模型表现出较优的性能.

### 参考文献

[1] M Herlihy, J Eliot, B Moss. Transactional memory: architectural support for lock-free data structures [A]. Proceedings of the 20th International Symposium on Computer Architecture [C]. San Diego: IEEE Computer Society, 1993. 289 – 300.

[2] L Hammond, V Wong, M Chen, et al. Transactional memory coherence and consistency [A]. Proceedings of the 31st International Symposium on Computer Architecture [C]. Munich: IEEE Computer Society, 2004. 102 – 113.

[3] C S Ananian, K Asanovic, B C Kuszmaul, et al. Unbounded transactional memory [J]. IEEE Micro, 2006, 26(1): 59 – 69.

[4] K E Moore, J Bobba, M J Moravan, et al. LogTM: log-based transactional memory [A]. Proceedings of the 12th International Symposium on High-Performance Computer Architecture [C]. Austin: IEEE Computer Society, 2006. 258 – 269.

[5] J Chung, L Yen, S Diestelhorst. ASF: AMD64 extension for lock-free data structures and transactional memory [A]. Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture [C]. Atlanta: IEEE Computer Society, 2010. 39 – 50.

[6] M Lupon, G Magklis, A Gonz′alez. A dynamically adaptable hardware transactional memory [A]. Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture [C]. Atlanta: IEEE Computer Society, 2010. 27 – 38.

[7] 窦强,王勇.事务存储系统中 PGHB 冲突检测算法改进 [J].电子学报,2010,38(1):195 – 198.
Dou Qiang, Wang Yong. The improvement of PGHB conflict detection algorithm in transactional memory systems [J]. Acta Electronica Sinica, 2010, 38(1): 195 – 198. (in Chinese)

[8] N Shavit, D Touitou. Software transactional memory [A]. Proceedings of the 14th annual ACM symposium on Principles of Distributed Computing [C]. Ottawa: ACM Press, 1995. 204 – 213.

[9] B Saha, A R Adl-Tabatabai, R L Hudson, et al. McRT-STM: A high performance software transactional memory system for a multi-core runtime [A]. Proceedings of the 11th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming [C]. New York: ACM, 2006. 187 – 197.

[10] R Ravi, J R Goodman. Transactional lock-free execution of lock-based programs [J]. ACM Operating Systems Review, 2002, 36(5): 5 – 17.

[11] A R Adl-Tabatabai, B T Lewis, V Menon, et al. Compiler and runtime support for efficient software transactional memory [A]. Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation [C]. Ottawa: ACM, 2006. 26 – 37.

[12] V Gramoli, R Guerraoui, V Trigonakis. TM2C: A software transactional memory for many-cores [A]. Proceedings of ACM European Conference on Computer Systems [C]. New York: ACM, 2012. 351 – 364.

[13] S Kumar, M Chu, C J Hughes, et al. Hybrid transactional memory [A]. Proceedings of the 11th ACM SIGPLAN symposium on Principles and practice of parallel programming [C]. New York: ACM, 2006. 209 – 220.

[14] A McDonald, B Carlstrom, J W Chung, et al. Transactional memory: The hardware-software interface [J]. IEEE Micro, 2007, 27(1): 67 – 76.

[15] J Bobba, N Goyal, M D Hill, et al. TokenTM: Efficient execution of large transactions with hardware transactional memory

[A]. Proceedings of the 35th International Symposium on Computer Architecture [C]. Beijing: IEEE, 2008. 127 – 138.

[16] 王绍刚,吴丹,庞征斌,等. HybridTCache:一种基于专用事务 Cache 的软硬件协同事务内存系统[J]. 计算机学报,2008,31(11):1907 – 1917.
Wang Shaogang, Wu Dan, Pang Zhengbin, Yang Xiaodong. HybridTCache: Tightly coupled hybrid transactional memory system to support efficient unbounded transactions with strong isolation [J]. Chinese Journal of Computers, 2008, 31(11): 1907 – 1917. (in Chinese)

[17] T Harris, A Cristal, O Unsal, et al. Transactional memory: An overview [J]. IEEE Micro, 2007, 27(3): 8 – 20.

[18] 彭林,谢伦国,张小强. 事务存储系统[J]. 计算机研究与发展,2009,46(8):1386 – 1398.
Peng Lin, Xie Lunguo, Zhang Xiaoqiang. Transactional memory system [J]. Journal Computer Research and Development, 2009, 46(8): 1386 – 1398. (in Chinese)

[19] M J Moravan, J Bobba, K E Moore, et al. Supporting nested transactional memory in LogTM [A]. Proceedings of 12th International Conference on Architectural Support for Programming Languages and Operating Systems [C]. New York: ACM, 2006. 359 – 370.

[20] P S Magnusson, M Christensson, J Eskilson, et al. Simics: A full system simulation platform [J]. IEEE, 2002, 35(2): 50 – 58.

[21] M M K Martin, D J Sorin, B M Beckmann, et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset [J]. Computer Architecture News (CAN), 2005, 33(4): 92 – 99.

[22] C C Minh, J W Chung, C Kozyrakis, et al. STAMP: Stanford transactional applications for multi-processing [A]. Proceedings of the IEEE International Symposium on Workload Characterization [C]. New York: IEEE, 2008. 35 – 46.

## 作者简介

**刘 轶** 男,1968 年出生于青海西宁,1990 年、1993 年和 2000 年在西安交通大学分别获得工学学士、硕士和博士学位. 现为北京航空航天大学计算机学院教授,博士生导师. 主要从事计算机系统结构、高性能计算等方面的研究.
E-mail: yi.liu@jsi.buaa.edu.cn

**吴名瑜** 女,1986 年出生于四川,北京航空航天大学计算机学院硕士研究生,主要从事计算机系统结构等方面的研究.
E-mail: mingyu.wu@jsi.buaa.edu.cn