

# HybridTCache: 一种基于专用事务 Cache 的 软硬件协同事务内存系统

王绍刚 吴 丹 庞征斌 杨晓东

(国防科学技术大学计算机学院 长沙 410073)

**摘 要** 文中提出一种高效的软硬件协同事务内存系统 HybridTCache. 在通常情况下, 事务完全由硬件执行, 当事务大小超出了硬件限制时, 操作系统将协同硬件执行. HybridTCache 提出了一种新的专用事务 Cache, 称为 TCache, 缓存事务执行过程中的临时数据, 由操作系统协同管理 TCache 溢出. 文中给出了基于 GEMS 模拟器的 HybridTCache 原型系统. 系统的评测显示 HybridTCache 比传统系统在性能、可扩展性、设计复杂度方面有较好的改进.

**关键词** 事务内存; TCache; HybridTCache; 软硬件协同

**中图法分类号** TP316 **DOI 号:** 10.3724/SP.J.1016.2008.01907

## HybridTCache: Tightly Coupled Hybrid Transactional Memory System to Support Efficient Unbounded Transactions with Strong Isolation

WANG Shao-Gang WU Dan PANG Zheng-Bin YANG Xiao-Dong

(School of Computer Science, National University of Defense Technology, Changsha 410073)

**Abstract** The paper proposes an efficient and unbounded hybrid-mode TM system, called HybridTCache, which maximum system performance by executing transaction completely in hardware in most common cases and triggering operating system (OS) support with low overhead when transaction size exceeds hardware capacity. HybridTCache provides new data version management by buffering transactional data in an independent cache, named TCache, which hides its value from other threads until transaction commits. TCache provides fast bookkeeping compared with traditional software approach, and makes both commit and abort fast. A key design point of hardware TM is to support unbounded transaction. HybridTCache achieves this by introducing TCache overflow exceptions and resorting OS to handle overflowed blocks. Evaluating the splash2 and STMAP benchmark suits shows that HybridTCache has advantages over traditional systems on performance, scalability and design complexity issues.

**Keywords** transactional memory; TCache; HybridTCache; hybrid-mode

## 1 引 言

在传统并行编程模型下, 并行应用程序需要使

用锁同步对共享资源进行访问, 其编程复杂、易产生死锁并且调试困难, 一直是并行编程领域有待解决的难题之一. 随着多核处理器技术的快速发展, 并行硬件平台已经成为各种应用的主流平台, 对并行编

收稿日期: 2008-05-31; 最终修改稿收到日期: 2008-09-08. 本课题得到国家自然科学基金(60803040, 60633050)资助. 王绍刚, 男, 1979 年生, 博士研究生, 研究方向为并行计算机体系结构、高性能存储系统. E-mail: wshaogang@nudt.edu.cn. 吴 丹, 女, 1979 年生, 博士, 讲师, 研究方向为新型体系结构、并行编译. 庞征斌, 男, 1972 年生, 博士, 副教授, 主要研究方向为高性能计算机体系结构、专用集成电路与 SoC 等. 杨晓东, 男, 1936 年生, 教授, 博士生导师, 主要研究领域为高性能计算机体系结构、分布与并行处理及 RAS 技术.

程的需求越来越大,传统基于锁机制的并行编程模型的缺点也越来越突显.事务内存(TM)正是在这种背景下提出的解决并行编程难题的技术之一<sup>[1-2]</sup>,近几年成为了学术界研究的热点.它为程序员提供了一种简洁、一致、无死锁的共享资源访问界面.在事务内存编程模型下,并行程序需要访问共享资源时,程序员只需要将访问共享资源的代码标注成一个原子执行的程序块(称为事务),对程序员透明的事务内存系统保证了事务执行时的原子性和隔离性.虽然事务内存是一项较新的编程模型,相关研究还没有十分成熟,但随着多核处理器技术的发展,事务内存技术已经成为并行计算领域最有前景的技术之一<sup>[3]</sup>.

事务内存系统执行事务的主要思路是系统提供事务间的冲突检测机制,当事务执行产生冲突时,即并发执行的事务同时访问了同一个共享资源,并且至少有一个是写操作,导致事务无法全部执行成功<sup>[4]</sup>,系统根据策略选择其中一个事务继续执行,而将其它事务作废,并恢复该事务执行前的系统状态.作废的事务将重新执行,直到执行成功为止.

从实现的角度看,目前研究所提出的事务内存系统可分成三大类:软件事务内存系统(STM)、硬件事务内存系统(HTM)和软硬件协同事务内存系统(HybridTM)<sup>[3]</sup>.STM系统大部分基于库函数或操作系统实现,是在当前硬件平台下支持事务内存编程模型的有效途径之一,但与细粒度的锁相比,其执行的开销较大、性能较低<sup>[5-7]</sup>.HTM系统完全由硬件执行事务,具有性能上的优势,缺点是对现有硬件体系结构有较大的改动,通常很难支持任意大小

的事务<sup>[8]</sup>.HybridTM系统的设计思路是由硬件执行系统的一部分功能,或者一些较小的事务,而对于一些较大的事务,交由软件来完成. HybridTM 具有实现相对容易、资源开销小、可扩展性强等优点,是非常重要的事务内存系统设计途径<sup>[9-11]</sup>.

本文基于已有的研究,提出了一种高效的软硬件协同事务内存系统 HybridTCache,支持并行程序中任意大小的事务(Unbounded)<sup>[11-12]</sup>及强隔离性(Strong-Isolation),主要创新点包括:(1)提出了专用事务 Cache 的体系结构(称为 TCache)缓存事务执行中的临时数据.事务执行过程中存储器的访问通过 TCache 完成,与传统的采用数据 Cache (DCache)缓存事务数据的方法相比,TCache 大大提高了事务内存系统执行事务的性能,并简化了系统设计.(2)提出了新的软硬件协同机制处理事务溢出.大部分情况下,事务完全由硬件执行.当缓存溢出时,TCache 控制器触发异常,由操作系统异常处理程序将溢出块保存在内核空间.这种机制的优点是软硬件之间高效协同,避免了传统设计中普遍存在的日志开销.(3)提出了支持事务内存的操作系统设计,包括执行模型、系统调用、中断异常处理、数据结构等.

2 系统概述

本节给出了 HybridTCache 系统的总体结构图,主要由硬件、操作系统、用户编程界面三部分组成,如图 1 所示.

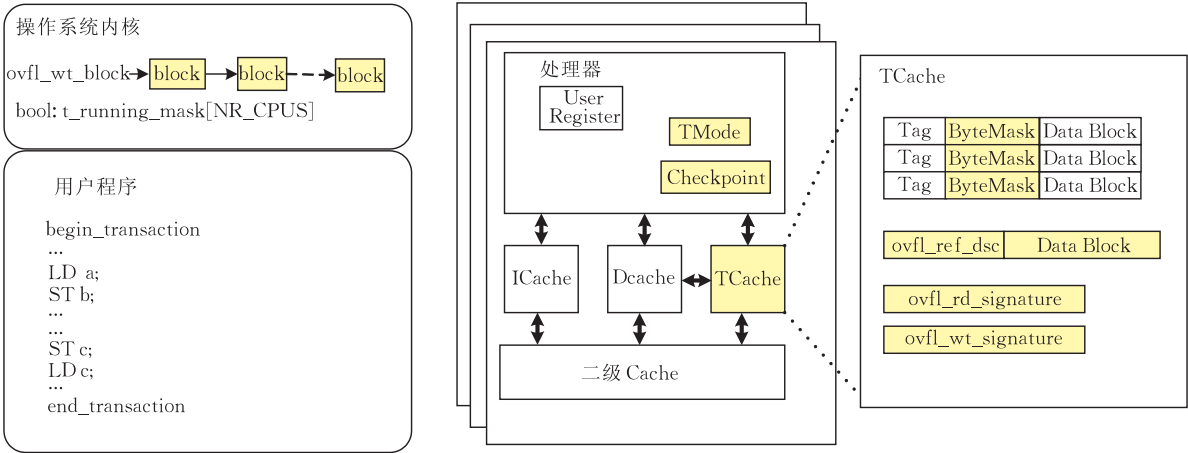


图 1 HybridTCache 系统结构图(阴影部分表示对现有体系结构的扩展)

传统硬件事务内存系统大多采用数据 Cache 缓存事务的临时数据,在这种方案下,事务块经常被

普通的块替换出 Cache,大大增加了事务执行时硬件溢出的概率<sup>[13]</sup>. HybridTCache 引入了专用事务

Cache,即 TCCache. TCCache 作为专用事务缓存连接在 L2 Cache 和处理器之间,暂存事务执行过程中的临时数据.事务还未提交前,TCCache 中的值对其它节点不可见;事务提交时,TCCache 通过 Flush 操作,将事务修改过的值提交到下级存储器中(L2);事务作废时,只需要将 TCCache 中数据作废,因而 HybridTCCache 同时提高了提交和作废的速度.

如何处理 HTM 和 HybridTM 系统中的硬件溢出一直是事务内存系统实现的难点.传统的方案是当 Cache 块被替换时,控制器将替换出的块更新到下一级存储器,但由于事务还未提交,更新的值可能是错误的,系统需要维护更新日志(log):事务在更新共享变量前,由软件保存变量的初始值,如果事务执行失败,则根据日志恢复存储器的初始值<sup>[13-14]</sup>.采用日志处理溢出的方式开销非常大. HybridTCCache 提出了一种新的由软件管理的事务块溢出机制.当 TCCache 中的块溢出时,控制器将触发相应的溢出异常.操作系统异常处理程序在内核中申请空间,保存溢出的块. HybridTCCache 采用了新的操作系统与硬件协同处理硬件缓存溢出的方式,不再需要日志,大大降低了系统的开销.

HybridTCCache 执行事务的思路是:当事务比较小,未引起 TCCache 溢出,事务完全由硬件执行;当事务较大,事务访存指令由于 TCCache 溢出而无法完成时,TCCache 控制器触发相应的异常,操作系统异常处理程序为该指令准备执行现场.异常处理程序执行完成后,引起异常的指令重新执行.

HybridTCCache 系统支持用户级事务,即在事务中的用户态 LD/ST 指令才被当成是事务访存指令,MMU 将通过 TCCache 访问存储器(如图 2 所示).当事务在执行过程中,由于中断异常处理、进程切换等操作而转入操作系统内核态或切换到其非事务线程时,处理器退出事务模式,通过 DCache 访问存储器.

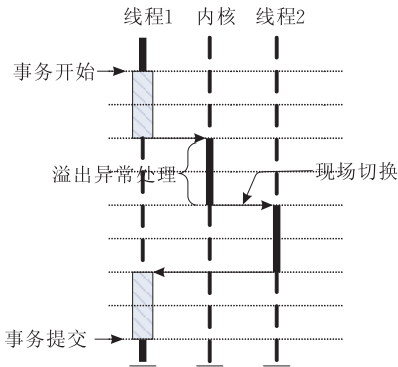


图 2 多线程环境下事务执行轨迹  
(阴影部分表示系统正处于事务状态)

### 3 HybridTCCache 硬件体系结构

本节主要介绍 HybridTCCache 硬件体系结构,其主要包括 TCCache、寄存器扩展、指令支持、硬件中断异常定义,分别在以下介绍.

#### 3.1 专用事务 Cache 体系结构

TCCache 中块的状态与数据 Cache 一致(共享 S 态、独占 M 态、空闲 I 态),在此基础上,为每个 TCCache 块设置 *ByteMask* 域,标识块中事务访问的具体字节.硬件事务内存设计需解决的一个重要问题是,如何检测和仲裁事务间冲突,与 LogTM 系统类似, HybridTCCache 采用基于目录的 Cache 一致性协议进行冲突检测和仲裁<sup>[13]</sup>.由于引入了 TCCache,协议需要增加相应的机制解决与传统数据 Cache 的交互问题,将在第 5 节介绍.

当数据 Cache 产生溢出时,如果被替换的块是“脏”块(被修改过的块),则需要将其写入到下一级存储器. TCCache 的情形有所不同,由于 Cache 块的值是推断的结果,如果更新到下一级存储器,则系统必须维护更新日志,以便事务执行失败时恢复存储器的状态. HybridTCCache 没有将 TCCache 替换出的“脏”块写入到下一级存储器,而由控制器触发异常,由异常处理程序将溢出块写入操作系统内核空间.为此,TCCache 中增加了两个签名寄存器: *ovfl\_rd\_signature* 和 *ovfl\_wt\_signature*. *ovfl\_rd\_signature* 记录 TCCache 中已经被替换的“事务读”(S 态)块地址集合, *ovfl\_wt\_signature* 记录替换的“事务写”(M 态)块地址集合.签名寄存器的作用是处理被替换出 TCCache 的块与其它节点的一致性冲突,并判定当前指令是否访问了溢出的 TCCache 块.签名寄存器是粗略表示地址集合的硬件结构,支持新地址插入签名和检测地址是否包含在签名中的操作.之所以称其是“粗略表示”,主要体现在:在对未插入过集合的地址检测是否包含时,可能返回包含的结果,称之为“伪命中签名”<sup>[14-15]</sup>.

#### 3.2 寄存器设计

HybridTCCache 引入新的寄存器,如表 1 所示.

表 1 HybridTCCache 寄存器设置	
寄存器	说明
checkpoint	保存 CPU 寄存器现场
Tmode	CPU 处于事务状态标志
ovfl_ref_desc	TCCache 溢出异常请求描述符寄存器
ovfl_wt_block	保存 TCCache 中溢出块的内容

处理器通过相应的指令(XStart)进入事务模式,置 Tmode 标识寄存器为 1.在事务处理模式

下,用户态的 LD/ST 指令通过 TCache 访问存储器.处理器执行事务之前,需要备份当前寄存器状态,保存在 checkpoint 寄存器中,执行事务失败时,从 checkpoint 中恢复寄存器状态,重新执行事务.

ovfl\_ref\_desc 和 ovfl\_wt\_block 寄存器是软硬件协同处理 TCache 溢出的“中介”,ovfl\_ref\_desc 寄存器保存 TCache 溢出异常信息(如表 2 所示),ovfl\_wt\_block 保存溢出块的内容.

表 2 ovfl\_ref\_desc 寄存器域定义

ovfl_ref_desc 域	位数/bit	来源	描述
rqt_valid	1	TCache	TCache 溢出请求有效
op	2	TCache	包括以下 3 种类型请求: add: TCache“脏”块溢出 update: ST 命中 ovfl_wt_signature rd: LD 命中 ovfl_rd_signature
phy_addr	32	TCache	溢出事务访存的物理地址
byte_mask	32	TCache	事务访问字节掩码
hit	1	OS	访存命中已经溢出的块

3.3 处理器指令集设计

HybridTCache 要求处理器增加相应的指令支持,如表 3 所示. XStart 和 XEnd 指令标识事务开始和结束,当处理器处于事务执行模式时,来自用户态的 LD/ST 指令被视为事务访存操作,经由 TCache 访问存储器.当一致性协议检测到冲突,需作废某个

事务时,TCache 控制器通过中断通知处理器,中断处理程序通过 XAbort 指令恢复由 XStart 指令保存的处理器现场,并通过 TDrop 指令作废 TCache 中的数据及状态.事务提交时,TFlush 指令将 TCache 中的修改过的数据更新到下级存储器,使事务执行结果对其它处理器可见.

表 3 处理器为支持事务所扩展的指令集

指令	说明
XStart	事务开始指令,置 Tmode 为 1,当前寄存器状态写入 checkpoint 寄存器
XEnd	事务提交指令,清除 Tmode,checkpoint 寄存器
XAbort	事务作废指令,从 checkpoint 恢复事务开始时寄存器状态,清 Tmode
TDrop	TCache 复位指令,作废所有块,重置状态寄存器
TFlush	将 TCache 中内容提交到 L2 Cache
SetOvflHit	事务访存命中溢出块,置 ovfl_ref_desc 的 hit 域
SetOvflBlock R1	从地址 R1 处复制一个 Cache 块到 ovfl_wt_block 寄存器
RdOvflBlock R1	将 ovfl_wt_block 内容复制到地址 R1 处

HybridTCache 将溢出的“脏”块写入 OS 内核空间,如果该块被再次访问(地址命中 ovfl\_wt\_signature),TCache 控制器触发访问溢出块异常.OS 异常处理程序检查内核空间是否保存有该地址的块,如命中,则由 SetOvflHit 指令置 hit 标志,并将块内容写入 ovfl\_wt\_block 寄存器.异常返回后,引起异常的指令将重新执行,本次访存的数据将从 ovfl\_wt\_block 寄存器获得.

3.4 TCache 中断/异常定义

事务执行过程中,TCache 可能产生以下 4 种情况而需要引入操作系统协同处理,分别定义成相应的异常或中断,如表 4 所示. Ovfl\_addblk\_exc,

ovfl\_rd\_exc 及 ovfl\_update\_exc 定义了由于 TCache 溢出触发的异常,由事务访存指令同步触发,异常信息及数据由 TCache 控制器分别保存在溢出请求描述符寄存器 ovfl\_ref\_desc 和溢出块寄存器 ovfl\_wt\_block 中. Abort\_transaction\_irq 中断是由一致性协议检测到事务冲突时触发的异步中断,通知操作系统作废当前正在执行的事务.本文提出的基于异常的溢出管理机制具有如下优点: TCache 溢出可由软件管理;消除 TCache 嵌套溢出异常,操作系统在处理异常时处于内核态,访存将通过 DCache,不会产生新的 TCache 溢出,简化了系统设计;系统不需要为事务“脏”块保存初始值,提高了事务执行的效率.

表 4 HybridTCache 中断/异常定义

描述	触发条件	标识
TCache 中“脏”块被替换	TCache 满,需替换“脏”块	ovfl_addblk_exc
读 TCache 溢出块异常	块地址命中 ovfl_rd_signature	ovfl_rd_exc
写 TCache 溢出块异常	块地址命中 ovfl_wt_signature	ovfl_update_exc
作废当前事务中断请求	一致性协议检测到冲突	Abort_transaction_irq

4 操作系统设计

HybridTCache 中,操作系统管理溢出的事务“脏”块,保存在 `ovfl_wt_blk` 数据结构中.属于某个进程已溢出的事务“脏”块组成链表的结构,进程描述符中的 `ovfl_wt_blk_ptr` 域指向该链表,如图 3 所示.`ovfl_wt_blk` 数据结构保存溢出块物理地址(`Line_phy_addr`)、数据(`datablk`)、事务访问字节掩码(`Wt_byte_mask`).TCache 中使用的是物理地址,因而溢出块提交时,需反向映射到其在内核空间的虚拟地址,由操作系统写回存储器.

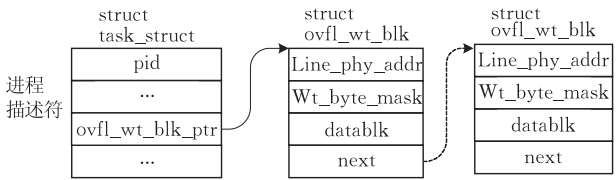


图 3 操作系统管理 TCache 溢出块的数据结构

在事务执行过程中,操作系统暂时将执行事务的线程绑定在某个处理器上:即事务执行时,不允许该线程迁移到其它节点上.为此,操作系统维护一个数组:`t_running_mask`,数组的大小等于系统处理器个数,以处理器的标识号为下标,表示该处理器当前是否被某个线程绑定.

源程序中通过 `Begin_Transaction` 系统调用标

识事务开始,通过 `Commit_Transaction` 系统调用提交事务.该系统调用的执行过程如图 4 所示.

事务开始
1. 将线程迁移到未绑定的处理器
2. 处理器通过 XStart 指令进入事务模式
事务提交
1. 处理器通过 XEnd 指令退出事务模式
2. 通过 TFlush 指令将 TCache 中值更新到存储器
3. 将保存在内核空间中的溢出块写回存储器

图 4 `Begin_Transaction` 及 `Commit_Transaction` 系统调用处理过程

HybridTCache 支持事务和非事务间的线程切换,非事务线程可以切换到绑定了事务的处理器上;事务线程只能被重新调度到其绑定的处理器上,事务执行完成后解除绑定.非事务线程访存通过数据 Cache,事务线程执行现场(TCache 状态)将保留,保证该线程被再次调度时正确执行.硬件事务内存系统支持线程切换是设计上的难题,HybridTCache 支持有限的线程切换,其实现简单、资源开销小.如果需要支持无限制的线程切换,可参照 LogTM-SE 系统的方案,但需要很大的开销<sup>[14]</sup>.

4.1 异常/中断处理流程

本节主要介绍 TCache 异常/中断的处理流程,包括 TCache 块溢出、CPU 访问 TCache 溢出“脏”块以及事务冲突三种需要软件协同处理的情况,如图 5 所示.

TCache 块溢出处理流程
1. CPU 发出的事物访存请求引发 TCache 块缺失,且不是访问已经被替换出的块(块地址不在 <code>ovfl_wt_signature</code> 中);TCache 没有空闲块,选择被替换的块;
2. 如果被替换的块是共享态,将块地址加入 <code>ovfl_rd_signature</code> ,转步 4;如果是独占态,置 <code>ovfl_ref_desc</code> 为块替换( <code>op=add</code> )请求信息,置 <code>ovfl_wt_block</code> 寄存器为被替换块的内容,控制器触发 <code>ovfl_addblk_exc</code> 异常;
3. 操作系统处理 <code>ovfl_addblk_exc</code> 异常,从 <code>ovfl_ref_desc</code> 和 <code>ovfl_wt_block</code> 寄存器中取溢出块信息,将其保存在内核空间中.异常退出后,CPU 重新执行引发块缺失的指令;
4. TCache 中已经腾出了一个空闲块,将请求发送给存储器.
CPU 访问 TCache 溢出“脏”块流程
1. CPU 发出事务访存请求引发 TCache 块缺失,但该块是以前被替换出的“脏”块(块地址命中 <code>ovfl_wt_signature</code> );
2. 如果是读请求,置 <code>ovfl_ref_desc</code> 为读请求( <code>op=rd</code> )信息,触发 <code>ovfl_rd_exc</code> ;写请求时,置 <code>ovfl_ref_desc</code> 为更新请求( <code>op=update</code> )信息,写入的值保存在 <code>ovfl_wt_block</code> 寄存器,触发 <code>ovfl_update_exc</code> 异常;
3. 如果是 <code>ovfl_rd_exc</code> 异常,操作系统查找 <code>ovfl_wt_blk_ptr</code> 链表,如果找到该地址的块,则将该块内容写入 <code>ovfl_wt_block</code> 寄存器,并置 <code>ovfl_ref_desc</code> 寄存器 hit 域为 1;如果是 <code>ovfl_update_exc</code> 异常,并且找到该地址的块,则用 <code>ovfl_wt_block</code> 寄存器中的内容更新该块,置 <code>ovfl_ref_desc</code> 寄存器 hit 域为 1;
4. 异常返回,触发异常的指令重新执行. TCache 控制器发现 hit 为 1,说明访问命中以前溢出的“脏”块.如果是读操作,将 <code>ovfl_wt_block</code> 中的数据返回给 CPU;如果是写操作,指令执行完成.
事务冲突处理流程
1. HybridTCache 通过一致性协议检测到当前执行的事务与其它节点产生冲突,控制器触发 <code>abort_transaction_irq</code> 中断,通知操作系统作废当前正在执行的事务;
2. 操作系统中断处理程序通过 <code>TDrop</code> 指令作废 TCache 中的内容,释放保存在内核空间的溢出块链表;
3. 通过 <code>XAbort</code> 指令从 <code>checkpoint</code> 寄存器中恢复处理器初始现场,重新执行事务.

图 5 TCache 异常/中断处理流程



## 5 一致性协议设计

本文提出的 Cache 一致性协议在 LogTM 系统的基础上,增加了对 TCache 的支持. 由于篇幅限制,仅介绍 HybridTCache 扩展的部分,原协议参见文献[13]. HybridTCache 使用基于目录的 Cache 一致性协议检测事务执行过程中与其它节点的冲突. 传统基于目录的 Cache 一致性协议检测事务冲突的大致流程是,当处理器 A 访问某个地址  $m$  时,由 Cache 控制器向存储器目录发出相应的一致性请求(共享读或独占读),处理器 A 得到响应后,置 Cache 块为共享态(共享读)或独占态(独占读);如处理器 B 也访问了地址  $m$ ,目录将 B 发出的访存请求转发给 A 处理器, A 接收到该请求后,将地址  $m$  的数据发送给 B,并发送消息到目录,更改地址  $m$  的目录状态<sup>[13,16]</sup>.

HybridTCache 检测事务间冲突的思路是:假设上述处理器 A 在事务内访问地址  $m$  并收到了目录的响应,在事务提交之前,处理器 A 暂时不响应由目录转发来的其它处理器的请求,从而保证了事务执行时的隔离性. TCache 控制器将溢出的“干净”或“脏”块地址分别保存在 `ovfl_rd_signature` 和 `ovfl_wt_signature` 中,如果目录转发给 A 的请求地址命中了 TCache 或溢出块地址集合(`ovfl_rd_signature`或`ovfl_wt_signature`), A 将向请求处理器发出 NACK 消息. 在执行事务过程中,如果 TCache 发出的访存请求收到 NACK 响应,说明事务与其它节点冲突,需要作废当前事务. 如果在执行事务过程中发生线程切换,切换前 TCache 的相关信息将保留,而且不会被新切换的线程破坏,硬件仍然能够处理该事务与其它节点的冲突.

程序在执行过程中,可能某个块同时被 DCache 和 TCache 请求(对共享变量的访问没有包含在事务中,或私有变量和共享变量被映射到同一个块), HybridTCache 存储控制器采用如下处理方式: (1) DCache 请求的数据从下级存储器获得. 如果 TCache 也有该块,或命中签名寄存器(`ovfl_rd_signature`或`ovfl_wt_signature`),且产生了冲突(R-W, W-W 冲突),则作废绑定在该处理器上的事务; (2) 如果 TCache 访问的数据块在 DCache 中,则从 DCache 中取得该块. 如果存在访问冲突(R-W, W-W 冲突),则写回并作废 DCache 中的块; 否则,从下级存储器获得.

为了降低程序员编程复杂度,并增强系统的

鲁棒性,事务内存系统一般要求支持强隔离性(Strong-Isolation)<sup>[9]</sup>,即系统可以正确处理事务与非事务访问间冲突. HybridTCache 通过以上的策略,支持 DCache 与 TCache 间的冲突检测及仲裁,保证了事务内存系统的强隔离性.

## 6 系统实现及评测

我们已经在 GEMS 模拟器<sup>[17]</sup>上实现了 HybridTCache 的原型系统. GEMS 模拟器是通用的多处理器系统模拟平台,在 simics 功能模拟器<sup>[18]</sup>的基础上,增加对存储系统和互联网络的模拟和评测功能. 目前提出的大部分硬件事务内存系统都基于 GEMS 模拟器进行性能评测,是目前事务内存领域被认可的性能评测平台. 平台选择 Serengeti 为目标机器,增加了 TCache 模块,设计了基于 MESI 的一致性协议管理处理器间及 TCache 的一致性,采用 simics 模拟器的 magic 指令实现了本文引入的新指令. 原型系统的参数设置见表 5.

表 5 HybridTCache 原型系统参数设置

参数	说明
处理器	16 个 UltraSparc-III-plus(单发射、顺序、IPC=1)
L1	8KB,独立指令和数据 Cache,4 路组相连,1 个周期延迟
TCache	2KB,4 路组相连,1 周期延迟,Flush 延迟 40 周期, Drop 延迟 10 周期
L2	1MB,4 路组相连,12 周期延迟
内存	4GB,100 周期延迟
网络	交叉开关,14 周期延迟
操作系统	Solaris10,异常/中断开销(周期):溢出 400,作废 1500
签名	采用 bloom 算法

本文选取了 splash2<sup>[19]</sup>和 STAMP<sup>[9]</sup>两种具有不同代表性的并行测试负载, splash2 是典型的多线程科学计算应用,需要频繁地同步共享变量的访问,由于被高度优化,共享资源的访问临界区一般都比较小,移植到事务内存编程模型后,相应的事务也比较小.

本文选取了其中的 barnes, cholesky, ocean-non-continuous, water-spatial, water-nsquare 和 radix 几个同步操作较多的程序. STAMP 是专用的事务内存测试集,源程序中包含了许多大的事务,本文选取了 STAMP 中的 vacation-low 和 vacation-high 测试程序, vacation-low 中大部分事务是只读的,因而事务冲突较少, vacation-high 中事务冲突的情况较多. 所有测试程序使用 SunStudio12 中的 CC 编译器,基于 POSIX 线程库编译.

6.1 性能分析

图 6 给出了在 HybridTCCache 和 LogTM 系统

下,访问共享资源的程序临界区的执行性能相比于使用锁同步情况下的加速比。

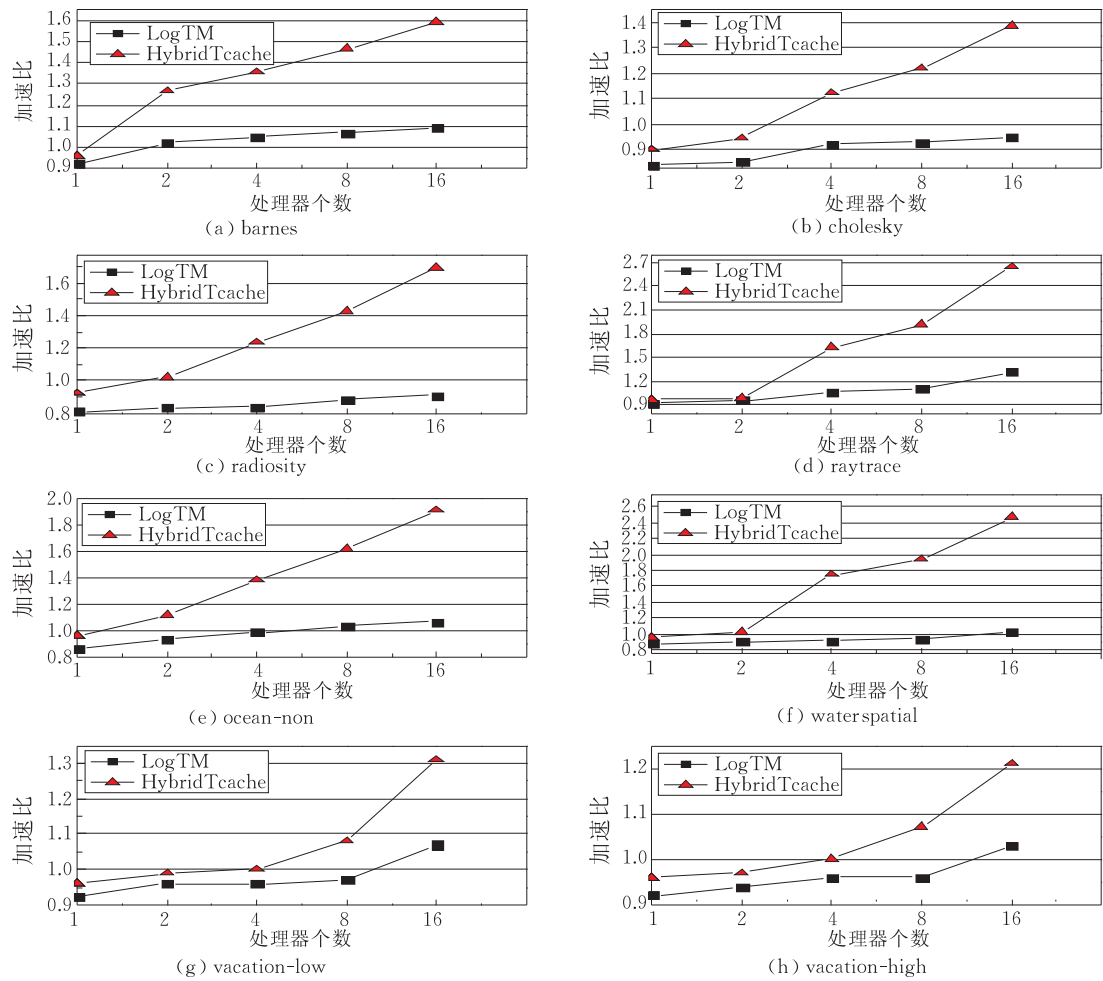


图 6 LogTM 和 HybridTCCache 相比于锁同步的性能加速比

从图 6 中可以看出,在测试小事务的程序时(barnes, cholesky, waterspatial, radiosity), HybridTCCache 相比锁同步机制显示出很高的加速比,比 LogTM 也有较明显的性能提升,此时, TCCache 溢出的情况很少,事务可完全由硬件完成,与 LogTM 相比,HybridTCCache 避免了由软件保存事务变量更新前初始值的复杂操作. HybridTCCache 采用的专用事务缓存提供了比 LogTM 系统更大容量的缓存,且减少了事务溢出概率,都有助于提高系统事务处理的性能. 从具体测试程序看, waterspatial 获得的加速比最高,原因是该程序中事务冲突较少,且事务中写操作比较多,更有利于发挥 TCCache 的性能.

当测试含有大事务的程序时(vacation-low, vacation-high), HybridTCCache 和 LogTM 系统相比于锁的性能加速比远小于执行 splash2 测试程序时获得的加速比,在 16 个处理器的 vacation-high

测试程序上, HybridTCCache 获得的加速比为 21%, LogTM 获得的加速比仅为 3%. 主要原因是事务较大时,大大增加了缓存溢出的概率,在溢出处理上花费的开销较大. 此外,系统的性能跟应用程序中事务的访问特点密切相关,事务中更新的变量越少,共享读的并行度越高,越有利于发挥 HybridTCCache 系统的性能.

6.2 可扩展性分析

相对于锁同步的共享资源访问机制,事务内存编程模型的优点之一是可扩展性好,允许共享资源的并行读,提高了应用的并行度. 从图 6 可以看出,处理器个数较少(小于 4 个)时, HybridTCCache 相比于锁并没有性能上的优势, HybridTCCache 的执行时间可能更长,主要原因是并行度较小时,事务重叠执行的概率较小, HybridTCCache 可扩展性的优势难以体现;而随着处理器数目的增多, HybridTCCache 显示出良好的可扩展性,例如,对 waterspa-

tial 测试程序,处理器个数达到 16 个时,HybridT-Cache 的执行性能是锁的 2.4 倍.从测试结果中可以看出,小事务应用的可扩展性要优于大事务应用的可扩展性.vacation-high 程序在 16 个处理器时的加速比为 1.2,比 1 个处理器时的加速比提高 50% 左右,而 16 个处理器的 waterspatial 程序的加速比比 1 个处理器时的加速比提高了 1.5 倍左右.

6.3 TCache 溢出分析

当 HybridTCache 由于溢出而转入操作系统处理时,事务执行的开销将大大增加,因而系统应尽量

避免 TCache 溢出.我们对 splash2 和 STAMP 测试程序执行时 TCache 的溢出情况作了统计,并与 LogTM 的溢出情况对比,如表 6 所示.本次测试中 TCache 容量为 2KB,LogTM 一级数据 Cache 容量为 4KB.从测试结果看,对于 splash2 测试程序,采用较小的 TCache 即可消除大部分的事务缓存溢出,而 LogTM 由于非事务访存的影响,事务块溢出的情况仍然比较多.而对 STAMP 测试程序,TCache 也明显降低了事务块溢出的概率,更有利于硬件性能的发挥.

表 6 splash2 及 STAMP 测试下 TCache 溢出情况统计

测试程序	事务数	访存总数	平均读次数	平均写次数	HybridTCache		LogTM
					溢出次数	访问溢出块次数	溢出次数
barnes	168	1328	6.1	4.2	2	8	129
water-ns	25	154	6.4	1.98	1	12	23
ray-trace	2432	13049	5.1	2.0	0	0	279
radiosity	12485	438545	1.8	1.5	6	4	381
ocean-con	59	426	5.6	0.13	0	0	107
cholesky	2389	10028	2.4	1.7	3	12	154
vacation-low	15437	1519032	70.7	18.1	483564	65777	956990
vacation-high	17285	1728659	99.1	18.6	529756	96165	1175488

为了分析 TCache 设计参数(容量、关联度)对系统性能的影响,我们测试了 16 个处理器的 vacation-high 程序在不同容量和关联度情况下相对于锁的加速比,如图 7 所示.从测试结果可以看出,对于大事务的程序,TCache 的配置对系统性能有比较明显的影响.如果 TCache 容量太小,系统花费在溢出处理的开销过大,其性能可能远低于使用锁同步的性能.在进行系统设计时,应根据负载中事务的访存特征,确定 TCache 的设计参数.

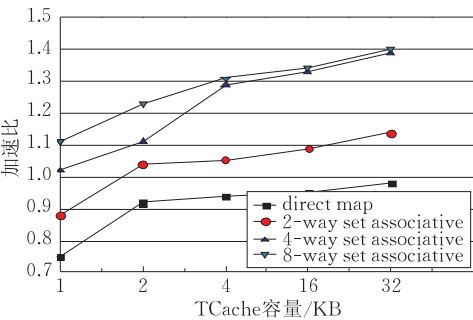


图 7 TCache 容量及关联度对系统性能的影响

6.4 签名算法分析

由于签名算法可能产生伪命中,带来一定的开销.为了研究算法对系统性能的影响,我们测试了几种签名寄存器配置下的性能,并与理想签名时的性能做比较.同时,我们也给出了 LogTM-SE 系统在相同配置下的性能变化情况,来比较 HybridT-

Cache 和 LogTM-SE 系统性能受签名算法影响的大小.我们选择的配置包括:2Kbit 的并行 H3 Hash 函数、4Kbit 并行 H3 Hash 函数和理想的签名(无伪命中),分别用 2KPH3, 4KPH3, PERFECT 表示,测试结果如图 8 所示.从测试结果可以看出,在小事务的测试程序上,HybridTCache 在 2KPH3 和 4KPH3 配置下的性能与理想签名算法情况下的性能相当.从上一小节统计 TCache 溢出的实验来看, splash2 测试下,TCache 几乎没有溢出,因而签名寄存器大部分情况下没有用到,很少的几次溢出没有产生伪命中,因而性能上没有损失. LogTM-SE 在 splash2 测试上,这三种配置下性能的差别也不大,但由于所有的冲突检测都需要访问签名寄存器,引入了一些伪冲突,因而性能有 5% 以内的损失.

在大事务测试程序上,LogTM-SE 和 HybridTCache 在不同配置下性能都有较大的差别.例如, vacation-high 测试下, LogTM-SE 在 2KPH3 和 4KPH3 配置下,性能分别是理想签名算法的 63% 和 89%,而 HybridTCache 分别为 76% 和 92%. HybridTCache 的性能受签名算法的影响比 LogTM-SE 要小,原因是 HybridTCache 仅有溢出块的地址需记录在签名寄存器,而 LogTM-SE 的所有事务块的地址都需要记录在签名寄存器,增加了伪命中的概率.



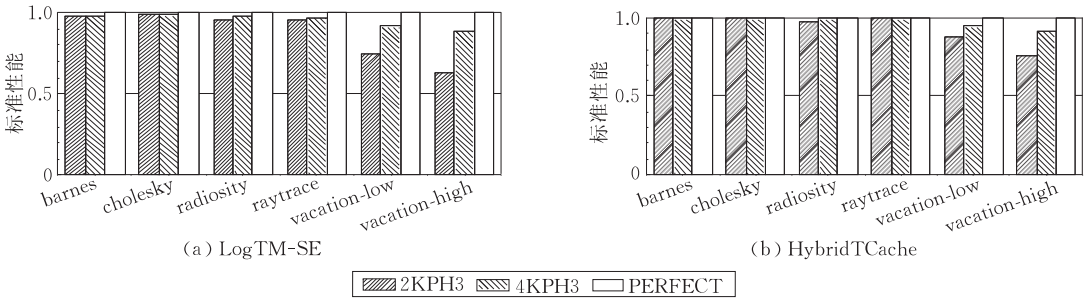


图 8 签名算法对 LogTM-SE 和 HybridTCCache 的性能影响

6.5 事务执行过程分析

基于对并程序访问共享资源特性的分析及已有的研究结果<sup>[20]</sup>, HybridTCCache 的设计哲学是, 事务执行时大部分情况下不存在冲突, 应尽量提高无冲突时事务执行的效率. 我们将 HybridTCCache 执行事务的过程分为“忙”(busy)、“提交”(commit)、“作废”(abort)和“溢出处理”(overflowed)4 种类型, 对各种类型操作的执行时间做了统计, 如图 7 所示, “忙”代表系统进行的是事务运算操作, 是“有用”的工作.

从图 9 中可以看出, 对于 splash2 这类事务较小的应用, 系统“忙”操作执行的时间占全部执行时间的 80% 以上, 说明 HybridTCCache 执行此类应用时的开销比较小.

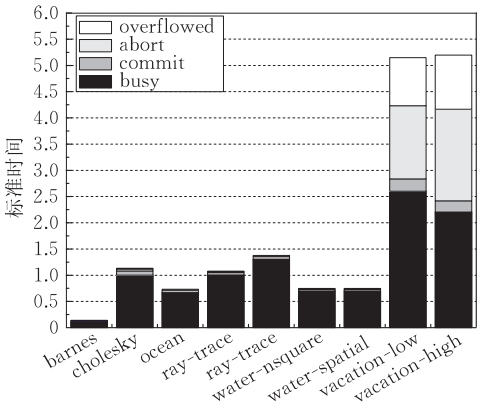


图 9 splash2 及 STAMP 测试下事务执行过程分析

当处理较大事务的应用时, 系统消耗在溢出处理和作废操作上的时间明显增加, HybridTCCache 再次访问缓存溢出“脏”块的开销远远高于访问未溢出块的开销. 同时, 大事务作废的代价也大大增加, 减少大事务间冲突对提高系统性能具有重要意义. STAMP 测试程序下, HybridTCCache 事务运算的时间占事务总执行时间的 60% 左右.

6.6 系统实现复杂性分析

与 LogTM 和 LogTM-SE 相比, HybridTCCache 减少的开销主要包括: 软件维护日志的开销、事务块

溢出所引发的“伪冲突”开销, 事务作废的开销; 而 HybridTCCache 的开销主要包括: TCCache 硬件开销、访问溢出事务块的异常处理开销. 从对 LogTM 及 HybridTCCache 系统的性能评测结果来看, HybridTCCache 所带来的开销要远小于 LogTM 系统执行事务时的开销. 处理器在 L1 级 Cache 中引入事务 Cache 所带来的硬件开销也较小; 从图 7 的 TCCache 容量相对性能的影响测试结果看, 小容量的 TCCache 就能带来比 LogTM 更高的性能. 已有的研究对经典的并程序分析表明, 大部分应用程序中访问共享资源的临界区很小, 在临界区中访问的共享变量数量也较少<sup>[20]</sup>, 因而事务执行过程中处理溢出中断的概率也较低; 并且系统在实现时, 可根据应用特征, 选择适当的 TCCache 参数, 实现系统最优配置.

由于事务 Cache 只负责处理事务访存请求, 处理器在执行非事务程序时, TCCache 将闲置, 带来了一定的资源浪费. 在资源受限的平台上, 与嵌入式系统所采用的策略类似<sup>[21]</sup>, 系统可采用可配置的 TCCache 设计, 在系统缓存总容量保持不变的情况下, 根据负载情况及当前运行模式, 可以对 TCCache 及 DCCache 的容量进行重配置; 当系统执行非事务程序时, TCCache 中的存储单元也可用来缓存非事务块的数据.

操作系统需要增加相应的中断、异常处理程序, 相对于操作系统其它机制相对独立, 其实现较为简单. LogTM 系统需要由软件保存事务变量更新前的初始值, 通过编译插入相关代码, 或由操作系统动态保存, 所需要处理的情况较复杂. 这类操作需要在非事务模式下完成, 要求系统支持事务线程与非事务线程间的现场切换. 但事务开始执行时, LogTM 系统屏蔽中断请求标志位, 暂时不响应外部中断. 因而降低了 LogTM 的实用性. 目前提出的支持进程切换和进程迁移的系统较少, LogTM-SE 是其中之一. 但其缺点是实现非常复杂, 并且会引入较多的“伪冲突”情况, 系统性能较低.

## 7 相关工作

随着多核处理器进入普及阶段,与之对应的软件技术成为新的研究热点,事务内存所具有的优点使其成为有希望解决目前编程难题的技术之一. 目前的研究已经提出了很多种软硬件结合的系统实现方案,如 LogTM<sup>[13]</sup>、LogTM-SE<sup>[11,14]</sup>、ONETM<sup>[11]</sup>等. 为支持任意大小的事务,这些系统必须在每次事务写之前由软件保存初始的数据,以便在事务执行失败时回退存储器的状态,当事务块被替换时,直接更新到存储器中. 这类系统即使在执行小的事务时,也需要软件操作. 已有的研究显示,大部分应用中的事务都比较小,因而与这些系统相比,HybridTCache 可以更有效地加速系统中出现概率高的情况.

UTM<sup>[22]</sup>系统也支持任意大小的事务,与大部分 HTM 系统不同的是,事务冲突检测不是基于 Cache 一致性协议,而由存储器新增的硬件逻辑完成. 事务的信息保存在一个很大的数据结构中,其中保存了 log 链表、事务状态(PENDING, COMMITTED, ABORTED)及其它信息. 系统为每个虚存块设置 R/W 标志位和指向对应 log 条目的指针. 如果事务块处在共享读状态,则 log 条目中保存了已访问该块的事务链表;如果事务块处在独占写状态,则 log 条目保存事务块初始值. 冲突检测基于事务块的状态. 与 HybridTCache 相比,UTM 的存储开销很大,硬件管理事务数据结构也带来很大的实现复杂度.

TokenTM<sup>[23]</sup>系统提出了基于令牌访问的事务冲突检测方法,系统为每个内存块增设了若干个访问令牌,访问内存单元之前应首先得到相应的访问令牌,读操作时需获取一个令牌,写操作要求获取所有令牌. 如果多次重试后无法获得令牌,则事务冲突,交由软件处理冲突. TokenTM 的优点是能够精确检测任意大小事务间的冲突,大的事务在执行过程中不会影响与它没有冲突的事务的性能,较好地支持进程切换、页交换等操作. TokenTM 的不足之处在于,令牌管理操作较复杂,尤其是令牌在处理器 Cache 中有多个副本时借贷和回收操作实现困难. 当溢出的事务作废或提交时,系统必须逐个释放令牌. 系统仍然需要由软件保存变量修改前的初始值.

目前提出的 HTM 系统绝大部分采用了将事务数据存储于数据 Cache 中的方案,根据前文所述,性能存在着瓶颈. 与本文专用事务缓存思想类似, TCC<sup>[24]</sup>也采用了专用的缓存保存事务执行时的临

时数据,但 TCC 采用广播的方式进行事务冲突检测,可扩展性不好. TCC 处理缓存溢出时,只能暂停事务执行,等待提交序号在它之前的事务提交后,才能继续执行. 因而, TCC 全系统只允许一个事务处于溢出状态,降低了系统的并行度.

VTM<sup>[25]</sup>系统也采用了将溢出的事务块保存在虚存空间的策略,并提供相应的机制处理已溢出的事务与其它事务之间的冲突检测及仲裁,支持事务线程切换等. 但其存在的问题是实现复杂,性能存在瓶颈. VTM 需要在 Cache 中保存块的虚拟地址,从虚拟 Cache 的研究来看,硬件设计将非常复杂. VTM 中溢出的 Cache 块保存在全局的列表数据结构中,进程下的所有事物线程共享该数据结构,在线程数目增多时,对该数据结构的并行访问将成为系统性能的瓶颈. VTM 只能依靠软件的方式检测溢出块间的冲突,非事务和事务访问不允许并行等方面都影响了系统性能.

## 8 结论及下一步工作

如果在执行事务过程中触发了非系统支持异常,并且异常处理程序需要访问用户空间的事务变量,系统将作废当前正在执行的事务而重新执行. 如何在系统中支持不可回退操作是事务内存系统设计普遍存在的问题之一,彻底解决这个问题还需要对事务执行的语义进行更深入的研究<sup>[26-27]</sup>,需要操作系统与硬件协同设计,将是本文下一步的研究工作.

综上所述,本文提出了一种高效能的操作系统与硬件紧密协同的事务内存系统,支持容量无限制的事务,并支持强隔离性. 据我们了解,本文提出的专用事务 Cache 以及基于异常的硬件事务内存系统溢出处理机制是首次提出的方案,为软硬件协同事务内存系统设计提供了一种新的思路. 在原型系统上进行的性能评测显示,HybridTCache 比传统的基于日志的事务内存系统有较好的性能提升,并具有良好的可扩展性.

## 参 考 文 献

- [1] Herlihy M, Moss J E B. Transactional memory: Architectural support for lock-free data structures//Proceedings of the Annual Symposium on Computer Architecture. San Diego, USA: IEEE, 1993: 289-300
- [2] McDonald A et al. Transactional memory: The hardware-software interface. IEEE Micro, 2007, 27(1): 67-76

- [3] Larus J R R, Ravi. Transactional memory. *Synthesis Lectures on Computer Architecture*, 2007, 2: 1-220
- [4] Spear M F et al. Conflict detection and validation strategies for software transactional memory//*Proceedings of the 20th International Symposium on Distributed Computing (DISC)*. Lecture Notes in Computer Science. Stockholm, Sweden; Springer Verlag, 2006
- [5] Marathe V J et al. Lowering the overhead of software transactional memory. University of Rochester; Technical Report 893, 2006
- [6] Jaswanth S et al. RSTM: A relaxed consistency software transactional memory for multicores//*Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*. Washington DC, USA, 2007; 428-435
- [7] Dice D, Shalev O, Shavit N. Transactional locking II. *Distributed Computing*, 2006, 4167: 194-208
- [8] Hammond L et al. Programming with transactional coherence and consistency (TCC)//*Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*. Boston, MA, United States, 2004; 1-13
- [9] Chi Cao M et al. An effective hybrid transactional memory system with strong isolation guarantees//*Proceedings of the 34th Annual International Symposium on Computer Architecture*. San Diego, USA, 2007; 69-80
- [10] Mark Moir, K. M. a. D. N. The adaptive transactional memory test platform: A tool for experimenting with transactional code for rock//*Proceedings of the 3rd ACM SIGPLAN Workshop on Transactional Computing*. Munich, Germany, 2008; 362
- [11] Colin B et al. Making the fast case common and the uncommon case simple in unbounded transactional memory//*Proceedings of the 34th Annual International Symposium on Computer Architecture*. San Diego, California, USA, 2007; 24-34
- [12] Weihaw C et al. Unbounded page-based transactional memory. *SIGOPS Operating Systems Review*, 2006, 40(5): 347-358
- [13] Moore K E et al. LogTM: Log-based transactional memory//*Proceedings of the International Symposium on High-Performance Computer Architecture*. Austin, TX, United States, 2006; 254-265
- [14] Luke Y et al. LogTM-SE: Decoupling hardware transactional memory from caches//*Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*. Phoenix, USA, 2007; 261-272
- [15] Martin D S et al. Implementing signatures for transactional memory//*Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. Illinois USA, 2007; 123-133
- [16] Chafi H et al. A scalable, non-blocking approach to transactional memory//*Proceedings of the International Symposium on High-Performance Computer Architecture*. Scottsdale, AZ, United States, 2007; 97-108
- [17] Milo M K M et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Computer Architecture News*, 2005, 33(4): 92-99
- [18] Peter S M et al. Simics a full system simulation platform. *Computer*, 2002, 50-58
- [19] Steven Cameron W et al. The SPLASH-2 programs: Characterization and methodological considerations//*Proceedings of the 22nd Annual International Symposium on Computer Architecture*. 1995; 24-36
- [20] Chung J et al. The common case transactional behavior of multithreaded programs//*Proceedings of the International Symposium on High-Performance Computer Architecture*. Austin, TX, United States, 2006; 266-277
- [21] Chuanjun Z, Frank V, Walid N. A highly configurable cache architecture for embedded systems. *ACM SIGARCH Computer Architecture News*, 2003, 31(2): 136-146
- [22] Ananian C S et al. Unbounded transactional memory//*Proceedings of the International Symposium on High-Performance Computer Architecture*. San Francisco, CA, United States, 2005; 316-327
- [23] Bobba Jayaram, N. G., Hill Mark D, Swift Michael M, Wood David A. TokenTM: Efficient execution of large transactions with hardware transactional memory//*Proceedings of the 33rd Annual International Symposium on Computer Architecture*. Beijing, China, 2008; 127-138
- [24] Hammond L et al. Transactional coherence and consistency: Simplifying parallel hardware and software. *IEEE Micro*, 2004, 24(6): 92-103
- [25] Ravi R, Maurice H, Konrad L. Virtualizing transactional memory//*Proceedings of the 32nd Annual International Symposium on Computer Architecture*. Madison, 2005; 494-505
- [26] Tim H. Exceptions and side-effects in atomic blocks. *Science of Computer Programming*, 2005, 58(3): 325-343
- [27] Chessell M et al. Extending the concept of transaction compensation. *IBM Systems Journal*, 2002, 41(4): 743-758



**WANG Shao-Gang**, born in 1979, Ph. D. candidate. His research interests include parallel computer architecture, parallel and distributed computing.

His main research interests include parallel compiler, software theory.

**PANG Zheng-Bin**, born in 1972, Ph. D. associate professor. His main research interests include high performance computer architecture, VLSI.

**YANG Xiao-Dong**, born in 1936, professor. His main research interests include high performance computer architecture, parallel and distributed computing.

**WU Dan**, born in 1979, Ph. D., associate professor.