第2期 2009年2月
电子学报 ACTA ELECTRONICA SINICA
Vol. 37 No. 2 Feb. 2009

一种支持事务内I/O操作的事务存储系统结构

刘 轶[1], 李明修[2], 张 昕[2], 李 鹤[2], 焦 林[2], 钱德沛[1,2]
(1. 北京航空航天大学计算机学院, 北京 100191; 2 西安交通大学 计算机系, 陕西西安 710049)

摘 要: 本文提出了一种支持事务内I/O操作的硬件事务存储系统结构. 该系统基于多核处理器结构和已有的cache一致性机制, 通过增加事务缓冲区和相关硬软件, 实现对事务的支持. 事务内I/O操作的实现基于事务提交锁的部分提交以及事务线程的阻塞/唤醒机制, 解决了事务内I/O操作所面临的回滚、事务迁移和缓冲区溢出等问题. 系统在模拟器中实现, 并利用5个测试程序对系统的性能进行了评价分析, 结果表明事务程序在系统中的性能相对于锁程序得到提升.

关键词: 事务存储; 多核处理器; 编程模型; I/O操作
中图分类号: TP302 文献标识码: A 文章编号: 0372-2112 (2009) 02-0248-05

Transactional Memory Architecture Supporting I/O Operations within Transactions

LIU Yi[1], LI Ming-xiu[2], ZHANG Xin[2], LI He[2], JIAO Lin[2], QIAN De-pei[1,2]
(1. *School of Computer, Beihang University, Beijing 100083, China;*
2. *Department of Computer, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China*)

Abstract: To support I/O operations inside transactions, this paper proposes a hardware transactional memory system architecture based on multi-core processor and current cache coherent mechanisms, It supports transactions by adding transactional buffer and related hardware and software. I/O operations within transactions are implemented by partial commit based on commit-lock, and blocking / waking-up of transactional threads. This solution solves or avoids the problems that I/O operations within transactions faced, including rollback, transaction migration and transactional buffer overflow. The system has been implemented by simulation. Its performance is evaluated by 5 benchmark applications. Simulation results show that the transactional programs executed in our system outperformed traditional lock-based programs.

Key words: transactional memory; multi-core processor; programming model; I/O operation

1 引言

随着多核处理器的快速发展, 传统并发程序难于编写且难于调试的问题日益突出. 在改善并行系统可编程性的研究中, 事务存储(Transactional Memory, TM)是一项引人注目的技术, 它不但可以改善并行系统的可编程性, 避免死锁, 还能够提高程序运行性能. 近年来随着多核处理器的发展, TM技术已成为计算机系统结构领域的研究热点之一. 在已有的硬件TM系统中, 对事务内I/O操作考虑较少, 已有系统结构均不能对其提供很好的支持.

本文针对硬件TM系统中的事务内I/O问题, 提出了一种硬件TM系统结构. 该系统基于多核处理器结构和已有的cache一致性机制, 通过增加事务缓冲区和相关硬软件, 实现对事务的支持. 事务内I/O操作则基于事务提交锁(commit-lock)的部分提交, 以及事务线程的阻塞/唤醒机制来实现. 系统在模拟器中实现, 并利用5个测试程序对系统的性能进行了评价分析.

2 事务存储及事务内I/O

2.1 事务存储技术简介

事务存储的概念首先在文献[1]中提出, 与传统的编程模型相比, TM技术的优点包括: 程序员只需进行事务的划分, 而不必过多地考虑线程间的同步/互斥, 因而改善了并发程序的可编程性; 事务间的互斥由系统自动完成, 避免了死锁的发生; 多个事务可以并发地投机运行, 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

收稿日期: 2008-01-15; 修回日期: 2008-08-10
基金项目: 国家自然科学基金(No. 60873053); 国家863高新技术研究发展计划(No. 2007AA01A127)

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

运行时性能得到提高.

根据实现方法的不同, 已有的事务存储系统结构可以被分为硬件 TM(Hardware Transactional Memory, HTM) 和软件 TM(Software Transactional Memory, STM). 硬件 TM 通过硬件来支持事务的原子性, 具有性能高、编程方便的优点. 缺点是需要对处理器及存储系统进行改动, 实现较为复杂, 并且受到缓冲大小的限制, 要求事务不能过大. 已有的硬件及硬软件混合事务存储系统结构主要有: TCC[2]、UTM/LTM[3]、LogTM[5] 等. 软件 TM 通过软件来支持事务的原子性, 可以在现有硬件平台上实现, 并且实现灵活、功能强大, 可以对事务提供更灵活的支持, 如事务的嵌套、部分回滚等. 缺点是由于事务管理和数据一致性维护的开销较大, 性能相对较低[7~10]. 鉴于硬软件 TM 各自的优缺点, 也出现了硬软件混合方案. 主要有: 通过虚拟化对程序员屏蔽硬件的特性的 VTM[11]、设置了硬、软件两种模式并自动切换的 Hybrid TM[12] 等.

2.2　事务内 I/O 问题及相关工作

对于硬件 TM 系统来说, 在事务内进行 I/O 操作将面临一系列问题, 主要体现为以下几点:

(1) 回滚问题: 多数 I/O 操作都不可撤消或很难撤消, 这使得在一个事务内执行 I/O 操作后, 该事务将很难进行回滚, 例如, 在一个事务内, 向文件中写入一条记录之后, 该事务需要回滚时将很难撤销这个记录.

(2) 事务迁移问题: I/O 操作时延较大, 操作系统会在 I/O 等待期间将线程阻塞, 操作完成后线程再次调度执行时可能被分配至另一处理器核, 而事务缓冲区的迁移很困难, 为了避免这一点, 很多硬件 TM 系统将事务长度限定在一个时间片内.

(3) 缓冲区溢出问题: I/O 操作通常涉及操作系统内多个模块的调用, 读/写操作可能较多, 容易引起事务缓冲区的溢出.

关于事务内 I/O 操作的支持, 在已有的硬件和硬软件混合系统结构中, 仅 TCC 在列举潜在改进时提出了在 I/O 操作前采用“伪溢出”方法, 获取事务的提交许可并保持至事务结束的思路. 它解决了上述 3 个问题中的回滚和缓冲区溢出问题, 但无法解决事务迁移问题, 而且 I/O 操作期间使其他事务提交时采取等待而不是睡眠的方法也会导致系统资源被过分占用.

3　支持事务内 I/O 操作的事务存储系统结构

3.1　事务存储系统结构

本文提出的支持事务内 I/O 操作的 TM 系统结构如图 1. 该系统基于现有的多核处理器结构, 并在处理器核内增加支持事务执行的硬件部件(图 1 中虚线框内部分). 处理器核的其他部分与传统处理器基本一致.

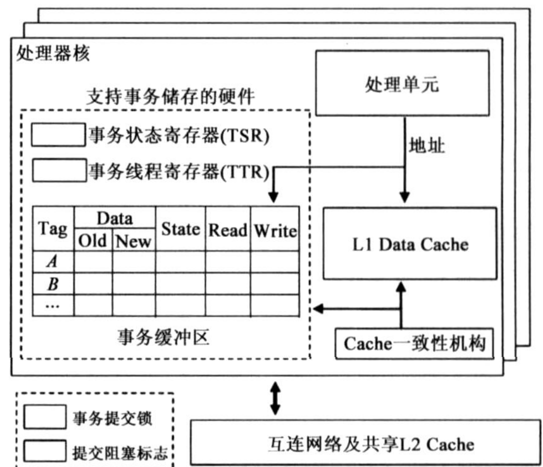

图1　事务支持硬件结构

支持事务存储的硬件以事务缓冲区为核心, 它主要用于缓冲事务执行过程中对变量所做的修改, 并保存相关状态, 以便最终提交或回滚. 其他硬件机构包括: 用于记录事务当前状态的事务状态寄存器(Transaction Status Register)、记录事务所属线程的事务线程寄存器(Transaction Thread Register)、事务提交锁和提交阻塞标志.

3.2　事务缓冲区结构

事务缓冲区结构与 cache 相似, 数据的存储也以行(line) 为单位, 主要差异在于, 对缓冲的数据保存新(new)、旧(old) 两个拷贝, 其中旧拷贝是指事务开始前的数据拷贝, 新拷贝则为事务执行过程中的修改版本. 每行数据还有 Read/Write 状态位, 用于记录该行数据是否被读或写过.

事务缓冲区与 L1 cache 处于同一层次, 处理器在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, 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被逐个清除, 使得新数据对 cache 一致性机构变为可见. 回滚操作时, 对事务缓冲区中每个被更新过的行, 将数据从 old 重新写回到 new 中.

3.3　事务执行过程

(1) 冲突检测与处理

分属不同线程的事务并发执行, 在两种情形下会发生冲突: 一个线程投机性读或写了一行数据, 随后另一线程试图写该行数据; 一个线程投机性写了一行数据, 其后另一线程试图读该行数据.

系统的冲突检测基于现有的 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, 其能够检测到上述情形的发生. 这样, 通过用硬件记录每个行是否被投机性读写过, 即可获知是否有冲突发生.

当检测到冲突后, 需进行冲突处理, 处理原则是: 如果冲突发生在事务和非事务代码之间, 则事务放弃

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

并回滚; 如果发生在两个事务之间, 则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

进行回滚时, 将事务缓冲区中所有投机性写过的行(Write 标志置位) 变为无效, 仅是投机性读过的行(仅 Read 标志置位) 不做变动, 随后将异常标志置位; 当本事务执行访存操作或提交时, 会产生一个异常, 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最终的提交操作, 该操作逐行清除事务缓冲区中的读/ 写标志位, 使数据对 cache 一致性部件可见.

4 事务内 I/ O 处理

4.1 基本思路

(1) 基于事务提交锁的部分提交

为保证事务的原子性, 并避免回滚, 让事务在 I/O 操作前进行部分提交(即提交事务缓冲区中的当前内容) , 并设置事务提交锁(commit-lock) 直到该事务结束. 事务提交锁保证系统中同一时刻只能有一个事务能进行提交操作, 即其他事务提交时必须等待.

部分提交后该 I/ O 事务的所有访存操作均不再进行缓冲, 即此时事务缓冲硬件被屏蔽. 这一方面提高了性能, 另一方面解决了 2.2 节中所述的事务迁移和缓冲区溢出问题. 这是因为: 首先事务在部分提交后事务缓冲区已被清除, 随后的访存操作也不再缓冲, 因此即使该事务因 I/O 被阻塞也无需恢复事务现场, 而且不会出现事务缓冲区溢出.

(2) 事务线程阻塞/ 唤醒机制

当事务提交锁置位时, 其他执行事务提交的线程均需等待, 为了不让其占用处理器资源, 将这些线程阻塞, 此时操作系统可调度其他线程运行.

当 I/O 事务结束并提交后, 唤醒因等待提交而阻塞的线程. 这些线程重新调度执行时, 将面临两种情形: 一是被调度到与阻塞前不同的处理器核上运行, 或事务现场已被破坏; 二是被调度到原来所在的处理器核上运行, 并且事务现场未被破坏. 对第一种情形, 线程清除当前事务现场, 并让事务重新执行; 对第二种情形, 事务继续进行提交操作.

该机制使得在 I/O 事务线程独占事务提交锁时, 系统可以阻塞后续要提交的事务线程, 并调度其他线程运行, 以提高处理器核的利用效率, 避免由于事务执行 I/O 操作导致程序运行停滞.

4.2 相关硬软件

为了实现事务内的 I/O 操作, 需要增加一定的硬件机构, 如表 1 所示.

表1 用于事务内 I/ O 处理的硬件机构

	宽度	配置方式	用途
事务提交锁	1bit	全局	标识当前是否有事务正在提交
提交阻塞标志	1bit	全局	标识当前是否有线程因等待提交而阻塞
事务状态寄存器(TSR)	≥3bit	每处理器核	记录当前事务所处状态: 事务/ 非事务、提交、放弃回滚等
事务线程寄存器(TTR)	= 线程 ID 宽度	每处理器核	记录当前事务所属线程 ID, 当事务阻塞后被重新调度执行时判断是否事务现场是否正常

除了硬件机构外, 还需以下配套程序:

(1) Commit_ lock_ exception_ handler

当事务进行提交时, 如事务提交锁置位, 在等待若干时钟周期后, 将触发该例程执行.

该例程将阻塞事务所在线程, 将其加入到提交等待队列中, 并将提交阻塞标志置位.

(2) Commit_ lock_ awake_ handler

当事务提交完成后, 如检测发现提交阻塞标志置位, 则触发该例程执行. 它将唤醒系统内所有因事务提交锁而阻塞的线程, 并清除提交阻塞标志.

以上处理程序可以作为操作系统的一部分进行实现, 从而保证对应用程序透明.

4.3 事务内 I/ O 处理

事务在进行 I/O 操作前, 将事务提交锁置位, 并进行部分提交, 之后的读/ 写操作不再缓冲; 在 I/O 操作执行过程中, 如线程被阻塞, 则调度其他线程运行, 由于事务缓冲区内容已被提交, I/O 事务不受影响; 在完成 I/O 操作后, 重新调度线程运行, 在新的处理器核中, 事务缓冲区为空, 且读/ 写操作直接针对主存, 因此最终事务结束提交时将成功完成.

当事务提交锁置位时, 如其他事务请求提交, 则重复检测该标志若干时钟周期, 在此期间如标志被清除则将标志置位并执行提交操作, 如超过时间限制, 则启动 commit_ lock_ exception_ handler, 它将阻塞该事务所在线程, 将其加入事务提交队列中, 并将阻塞标志置位.

在 I/O 事务结束并提交后, 如线程提交锁置位, 则执行 commit_ lock_ awake_ handler, 它将清除事务提交锁并唤醒在其上阻塞的线程. 这些被唤醒的线程重新调度执行时, 检测事务线程寄存器(TTR) 是否与本线程相符, 如相符, 事务继续进行提交操作; 否则清除当前事务现场, 并重新启动事务.

5 评价与分析

5.1 实验环境与实验方法

系统的性能评价基于系统结构模拟器 Virtutech

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

Simics 及多核扩展包 GEMS 进行, 通过对模拟器的扩展,实现了本文提出的事务存储系统. 目标系统基于 Sparc 处理器, 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, 运行 Solaris 操作系统. 表 2 给出了目标系统的硬件参数.

表2　目标机硬件参数

Cache 大小	L1: 16kB+ 16kB　L2: 4MB 事务缓冲区: 16kB
Cache line	64 bytes
一致性协议	MOESI 协议
互连网络	Hierarchical switch topology

使用5 个测试程序对目标系统进行了性能评价, 其中3 个为自编测试程序, 用于测试事务内 I/O 性能, 这3 个程序的 I/O 操作密集程度各不相同, 另外 2 个测试程序来自程序集 SPLASH-2.

表3　测试程序

程　序	来　源	说　　明	事务内 I/O
deque1	自编	双向队列入出+ 写文件	密集
deque2	自编	双向队列入出+ 写文件	普通
deque3	自编	双向队列入出+ 写文件	稀疏
FFT	SPLASH-2	快速傅立叶变换	无
LU	SPLASH-2	矩阵的 LU 分解	无

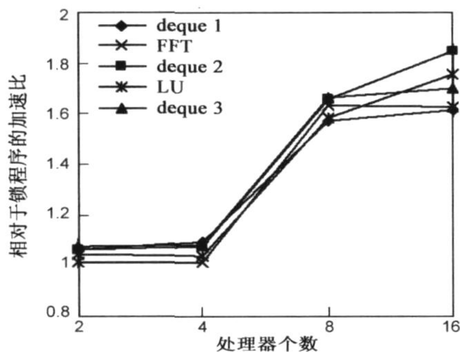

图2　事务程序相对于锁程序的加速比

5.2　实验结果及分析

图 2 给出了事务程序相对于锁程序的加速比, 可以看出, 随着处理器个数的增加, 事务程序相对于锁程序的性能提升也逐渐增大, 这是由于随着处理器的增多, 多个线程访问共享数据时的竞争关系更加激烈, 锁程序由于线程间互斥造成的阻塞相对增多, 而事务程序中访问共享数据的代码以事务的形式投机运行, 线程阻塞相对较少.

图 3 给出了事务程序和锁程序的阻塞时间之比, 可以看出, 事务程序的线程阻塞时间要比锁程序少得多, 当然, 由于事务程序存在事务冲突回滚的情形, 这部分减少的阻塞时间中仅有一部分能够体现在运行时间上. 需要说明的是, 由于程序 FFT 和 LU 中无 I/O 操作, 其事务程序为无阻塞运行, 因而图 3 仅给出了 3 个程序的测试数据.

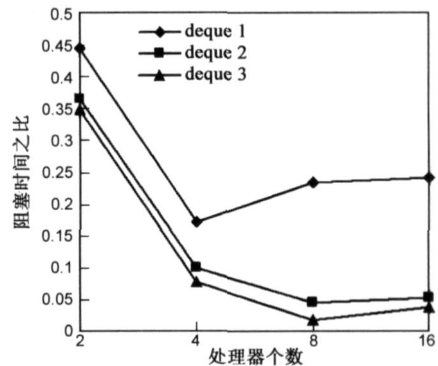

图3　事务程序与锁程序的阻塞时间比

图 4 给出了事务程序运行时启动事务个数与成功提交事务个数的比值. 一个事务由于冲突回滚后需要被重新启动, 因而该比值反映了事务回滚的频繁程度, 如该比值为 1 则表示无事务回滚发生, 比值越大说明事务回滚越频繁. 从图中可以看出, 该比值基本保持在相对较低的水平, 由于 deque1 为事务内 I/O 操作密集的

表4　事务程序运行时间的比例构成

处理器个数	时间指标类别	占总运行时间的比例(%)				
		deque1	deque2	deque3	FFT	LU
2	线程阻塞	4.924%	2.431%	1.005%	0.000%	0.000%
	事务处理	0.055%	0.056%	0.052%	0.033%	0.062%
	程序代码	95.021%	97.513%	98.942%	99.967%	99.938%
4	线程阻塞	3.037%	1.439%	0.947%	0.000%	0.000%
	事务处理	0.054%	0.052%	0.048%	0.020%	0.029%
	程序代码	96.908%	98.508%	99.005%	99.980%	99.971%
8	线程阻塞	14.902%	3.424%	0.305%	0.000%	0.000%
	事务处理	0.151%	0.156%	0.148%	0.046%	0.065%
	程序代码	84.946%	96.420%	99.546%	99.954%	99.935%
16	线程阻塞	15.085%	3.254%	0.325%	0.000%	0.000%
	事务处理	0.117%	0.152%	0.141%	0.039%	0.036%
	程序代码	84.798%	96.594%	99.535%	99.961%	99.964%

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

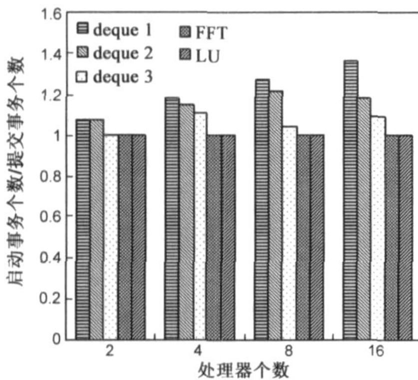

图4 启动事务数与提交事务数比

程序,事务执行时间相对较长,事务间冲突发生的概率也较大,因而比值相对较高.

表4给出了事务程序运行时间的比例构成.程序的总运行时间被分成三部分:线程因 I/O 操作或等待事务提交锁而阻塞的时间、事务启动/提交/回滚等处理开销、程序代码的实际执行时间.可以看出,事务处理开销在总运行时间中所占比例很小,而线程阻塞时间则与程序的事务内 I/O 操作密集程度相关,事务内 I/O 越密集,线程阻塞时间所占比例越大.

6 结论

对于事务存储系统来说,在事务内进行 I/O 操作是难以避免的.硬件TM 系统在事务内进行 I/O 操作时面临一系列问题,包括 I/O 操作后难以回滚、事务迁移和缓冲区溢出等.已有的硬件TM 系统结构对事务内 I/O 考虑较少,不能对其提供很好的支持.

本文提出了一种支持事务内 I/O 操作的硬件 TM 系统结构.该系统基于多核处理器结构和已有的 cache 一致性机制,通过增加事务缓冲区和相关硬软件,实现对事务的支持.事务内 I/O 操作通过两种机制来实现,即基于事务提交锁的部分提交,以及事务线程的阻塞/唤醒机制.这种硬件TM 系统结构解决了事务内 I/O 操作所面临的问题.基于模拟器实现了所提出的系统,并利用5 个测试程序对系统的性能进行评价分析,结果表明事务程序在系统中的性能相对于锁程序得到提升.

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 31th International Symposium on Computer Architecture Proceedings[C]. Munich: IEEE Computer Society, 2004. 102- 113.

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

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

[5] 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.

[6] 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.

[7] B Saha, A 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.

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

[9] T Harris, S Marlow, S P Jones, et al. Composable memory transactions[A]. Proceedings of 2005 ACM SIGPLAN Symposium on Principles and Practise of Parallel Programming[C]. Chicago: ACM, 2005. 48- 60.

[10] R Rajwar, M Herlihy, K Lai. Virtualizing transactional memory[A]. Proceedings of the 32th International Symposium on Computer Architecture[C]. Madison: IEEE Computer Society, 2005. 494- 505.

[11] 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.

作者简介:

刘 轶 男,1968 年生于青海西宁,博士,副教授,主要研究方向为计算机系统结构、计算机网络.
E-mail: yi.liu@jsi.buaa.edu.cn

李明修 男,1984 年生于湖北仙桃,硕士研究生,主要研究方向为计算机系统结构、计算机网络.

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net