# 面向 CC-NUMA 体系结构的事务内存冲突规避方法

王睿伯 卢锡城 卢 凯 王绍刚
(国防科技大学计算机学院 长沙 410073)

**摘 要** 现有的事务内存研究主要面向多核处理器和 SMP 机器,缺少对 CC-NUMA 系统的研究.而 CC-NUMA 是高端服务器的重要体系结构,随着用户对并行处理能力需求的不断上升,高端服务器将占据越来越重要的地位.文中概要阐述事务内存研究的基本情况,通过详尽的实验数据,深入分析了 CC-NUMA 结构的本地、远程访存差异特性对事务内存性能的影响,提出了一种面向 CC-NUMA 体系结构的冲突规避方法 PBC. PBC 在事务启动之前,对冲突可能性进行预测,并根据预测结果对事务进行调度,以降低事务的失败率.实验表明,文中提出的 PBC 方法可以显著提高 CC-NUMA 机器上运行事务内存的整体性能.

**关键词** 事务内存;CC-NUMA;冲突规避
**中图法分类号** TP393 **DOI 号**: 10.3724/SP.J.1016.2011.00676

## CC-NUMA Oriented Conflict Preventing Method for Transactional Memory

WANG Rui-Bo LU Xi-Cheng LU Kai WANG Shao-Gang
(*School of Computer Science, National University of Defense Technology, Changsha* 410073)

**Abstract** Transactional Memory (TM) research has focused on multi-core processors and SMP machines, leaving the area of CC-NUMA (cache-coherent Non-Uniform Memory Access) machines unexplored. CC-NUMA is the key architecture of high-end servers. With the increasing demand of parallel processing capabilities, the high-end servers are more important than ever before. We briefly introduced the research background of the TM, and analyzed the TM performance degradation from CC-NUMA's different local and remote memory access latency through detailed experiments. We present a conflict preventing method called PBC for the TM using on CC-NUMA architecture. The PBC method predicted the conflict probability before the start of a transaction, and scheduled the transactions by the predicted probabilities. The aim is to reduce the failure rate of transactions. Experiment results show that the PBC method can achieve better TM performance on CC-NUMA machines.

**Keywords** transactional memory; CC-NUMA; conflict preventing

## 1 引 言

随着多核技术的普及,处理器提供了充裕的硬件并行能力,而与之形成鲜明对比的是,现在的大多数软件还不是并行的.软件的并行编程技术一直"曲高和寡",并行编程作为高性能计算领域常用的编程模式,并没有普及到所有的编程人员,并且一直被当做是"专家级"的技术[1-2].目前,并行编程中关键的共享资源同步技术仍然在"锁"机制的基础上实现[3-6],通过对共享数据"加锁"访问来保证并行程序执行的正确性.事实上如何正确地使用锁是一件非

收稿日期:2010-08-26;最终修改稿收到日期:2011-01-03.本课题得到国家自然科学基金(60621003)资助.**王睿伯**,男,1981 年生,博士研究生,研究方向为操作系统、高性能计算. E-mail: ruibo@nudt.edu.cn. **卢锡城**,男,1946 年生,教授,博士生导师,中国工程院院士,主要研究领域为分布式处理技术、计算机网络与通信技术. E-mail: xclu@nudt.edu.cn. **卢 凯**,男,1973 年生,教授,博士生导师,主要研究领域为操作系统、高性能计算. **王绍刚**,男,1979 年生,博士,主要研究方向为并行计算机体系结构、高性能存储系统.

常困难的事，非常容易造成死锁[7]等问题.

在应用需求的驱动下，学术界已经开始进行了新的同步模型的研究，其中事务内存无疑是最具有吸引力和应用前景的一项新技术，它能够有效地解决并行编程当前所面临的难题，成为近年来的研究热点，也是并行计算机体系结构的前沿研究课题之一[8-11].

目前的事务内存主要面向单个多核处理器以及 SMP 机器，缺少对 NUMA 结构的关注，特别是 CC-NUMA（Cache Coherence Non-Uniform Memory Access）结构. 而 CC-NUMA 体系结构是当前高性能服务器的主流体系结构，例如 AMD 公司 Opteron 系列处理器的 HyperTransport 互联技术，还有 Intel 公司最新的至强和安腾处理器的 QuickPath 互联技术. NUMA 结构的特点，就是每个处理器访问本地和远程内存的延迟不同，访问远程内存的延迟往往是本地的若干倍或者更多，具体差异取决于处理器互联的层次、距离以及互联接口的带宽和延迟.

本文通过实验，深入分析了现有的事务内存系统在 CC-NUMA 结构机器上的运行情况，发现随着远程内存访问延迟的增加，事务之间的冲突开始增多，导致事务的失败率随之增加，极大地影响了系统性能. 由此，本文提出了面向 CC-NUMA 的事务冲突规避方法 PBC，该方法在事务开始之前，预测该事务将来发生冲突的可能性，根据预测结果对事务进行调度. 实验证明，该方法在 CC-NUMA 结构机器上可以有效地减少事务冲突，提高事务内存系统的性能.

## 2　相关研究

### 2.1　事务内存

事务的概念源自于数据库管理系统. 在数据库管理系统中，事务必须满足 ACID 性质，即原子性、一致性、隔离性和持久性. 用户可以使用事务来包装多个数据库操作，在其他用户看来，事务中的多个数据库操作一次性完成，事务执行过程的中间状态对其他用户不可见[12]. 将共享数据的访问对应于数据库操作，事务内存系统可以对应于数据库管理系统，用户用事务来包装多个共享数据的访问，由事务内存系统来保证事务特性.

在采用事务内存同步方式的程序中，一般使用关键词 atomic 将指令序列包装成事务，由事务内存系统保证共享数据的一系列操作是原子执行的. 可以看出，在事务内存同步模型下，程序员的工作非常简单：将需要互斥执行的一系列操作在源代码中标注为一个事务即可，而不需要考虑事务之间可能产生的具体冲突情况[13-14].

现有的事务内存系统执行流程如图 1 所示，事务开始后，系统将监测所有的共享数据访问，进行冲突检测，如果在事务完成时都没有发现冲突，则事务成功提交；如果发现冲突，则由竞争管理算法选择两个冲突事务的其中之一挂起或者重启.

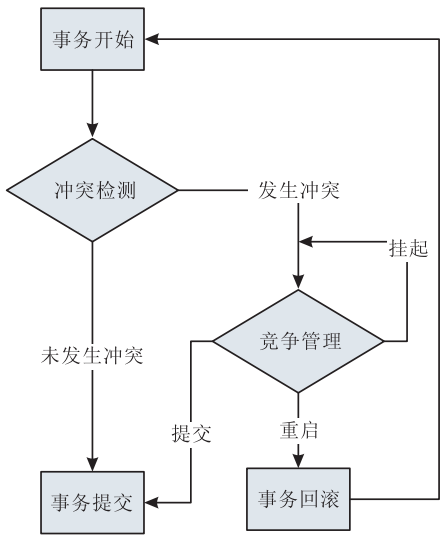


图 1　现有事务内存系统的事务执行流程

现有的事务内存研究大都面向多核处理器以及 SMP 机器，还尚无针对 CC-NUMA 体系结构的研究. 类似的研究主要针对分布式内存系统，例如集群. Manassiev 等人在 2006 年提出了一个针对集群设计的事务内存系统[15]，该系统的底层通过软件分布式共享内存（Software Distributed Shared Memory）TreadMarks[16]实现，TreadMarks 为集群提供了一个全局的共享内存视图，类似于将集群模拟成一台 SMP 机器，这种实现方式规避了集群的内存分布特性，仅仅提供了一种让事务内存在集群上运行的途径. 2008 年提出的 Cluster-STM[17]和 DiSTM[18]也都是针对集群设计的事务内存系统，它们没有使用分布式共享内存的机制来保证共享数据的一致性，而是自己实现各节点间的共享数据同步，分别使用了 GASNet[19]和 ProActive[20]库来进行内存远程访问. 其中，DiSTM 的设计面向小型集群，而 Cluster-STM 面向大规模高性能计算集群. 由于集群系统节点间的网络互联速度远远低于处理器芯片的内部互联速度，因此在集群上运行的事务内存系统性能都

比较差，系统运行过程中，各个节点间需要使用互连网络频繁交换大量数据以实现事务的冲突检测，相对于多核处理器、SMP 机器来说，这种通过远程网络通信进行的内存访问大大地降低了系统性能.

### 2.2 CC-NUMA 体系结构

目前很多主流的高性能服务器都采用了 CC-NUMA 体系结构，例如 AMD 公司的 Direct Connect 和 Intel 公司的 QuickPath 技术，还有 HP 的 Superdome 以及 SGI 的 Altix 服务器等等.

CC-NUMA 的系统结构如图 2 所示，每个处理器都配置了内存，此时处理器访问本地内存和远程内存的延迟存在差异，访问远程内存的延迟往往是本地的几倍甚至几十倍. 与集群系统不同的是，CC-NUMA系统配置了缓存一致性(cache coherent)硬件，使得这种机器在逻辑上呈现 SMP 特性，现有的事务内存系统可以不加修改地在上面直接运行.

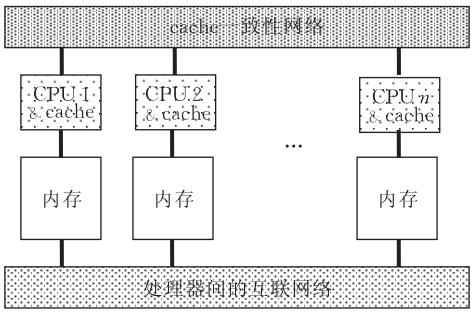


图 2 CC-NUMA 系统结构

## 3 已有事务内存机制在 CC-NUMA 机器上的实验分析

虽然现有的事务内存系统可以直接在 CC-NUMA机器上运行，但已经有实验表明，NUMA 特性对事务内存系统的性能有较大影响[21-22]. 为了分析原因，更好地比较事务内存在 SMP 机器与 CC-NUMA 机器上的运行情况，本文使用 Simics 模拟器进行了实验验证，模拟的机器拥有 4 个处理器，每个处理器拥有 4 个 1GHz 的 x86-64 核心并配置 4GB 内存. 这样，系统整体拥有 16 个处理器核心和 16GB 内存. NUMA 结构带来的访存延迟差异通过一个基于 GEMS[23] 构建的 Simics 模块实现. 使用模拟器进行实验，是为了更好地分析、对比 NUMA 的远程与本地访存延迟比值对系统性能的影响. 模拟参数设定如下，每个处理器核心拥有 64KB 的指令 cache 以及 64KB 的数据 cache，访问延迟均为 3 个时钟周期，二级 cache 的容量为 1MB，访问延迟为 16 个周期，本地内存的访问延迟为 240 个周期. 处理器之间全互联，远程内存访问延迟设定为本地的 2、4、8 倍. 模拟器上运行的操作系统为 Fedora 5，操作系统核心版本是 Linux 2.6.15.

测试的 Benchmark 选择了 STMBench7[24]，该测试程序足够复杂，比较接近真实应用. 本文选择了 3 个主流的事务内存系统进行实验：RSTM[25]、TinySTM[26] 和 TL2[27]. Benchmark 的可执行程序均使用 GNU g++编译，参数为－O3 和－DNDEBUG. 各个事务内存系统的编译均采用其缺省参数，所有的实验结果都是运行 10 次后得到的结果平均值. Stmbench7 参数配置为测试一个小规模的数据集，对数据结构的操作是读写并发的.

实验结果如图 3 所示，测试程序分别在线程数量 2、4、8、16 的情况下运行，实验结果为 Stmbench7 的平均事务吞吐量，单位是每秒的操作数量，数量越高表示性能越好. NUMA 1∶2、1∶4 和 1∶8 表示 CC-NUMA 机器的本地与远程访存延迟比值. 可以看出，在 SMP 机器上 TinySTM 的性能比较好，线程数为 8 时性能比 TL2 要高出 40％左右，在线程数为 16 时性能比其它两种事务内存均高出 30％左右. 而在 CC-NUMA 系统上情况有了变化，在 NUMA 1∶2 的情况下，RSTM 在线程数 8 时的性能超越了 TinySTM，不过在线程数 16 时仍然是 TinySTM 性能占优；而在 NUMA 1∶4 的情况下，线程数 16 时 3 种事务内存的性能基本相当，差别不大了. 到了 NUMA 1∶8 的情况，随着远程内存访问延迟的增加，3 种事务内存的性能都有明显的下降，其中受影响最大的是 TinySTM，在线程数 16 的情况下，性能比 SMP 时下降了 75％以上，而其它两种事务内存也下降了 70％左右. 可见，NUMA 结构特性对事务内存的性能有较大影响.

为了分析性能下降的原因，实验中采集了 3 种事务内存系统在线程数 16 时的运行时信息，包括冲突检测时间、事务提交时间以及事务处理之外的计算时间，将这些数据规范化之后的结果如图 4 所示. 图示中“Transaction Commit”表示事务提交占用的时间，“Conflict Detection”表示事务之间冲突检测占用的时间，“Computation”表示事务处理之外的正常计算代码执行时间.

可以看出，相对于SMP机器而言，在CC-NUMA机器上运行的事务内存系统自身开销较大. 以 RSTM 为例，在本地、远程访存延迟比为 1∶8 的情况下系统耗费了将近 80％的时间在冲突检测和事

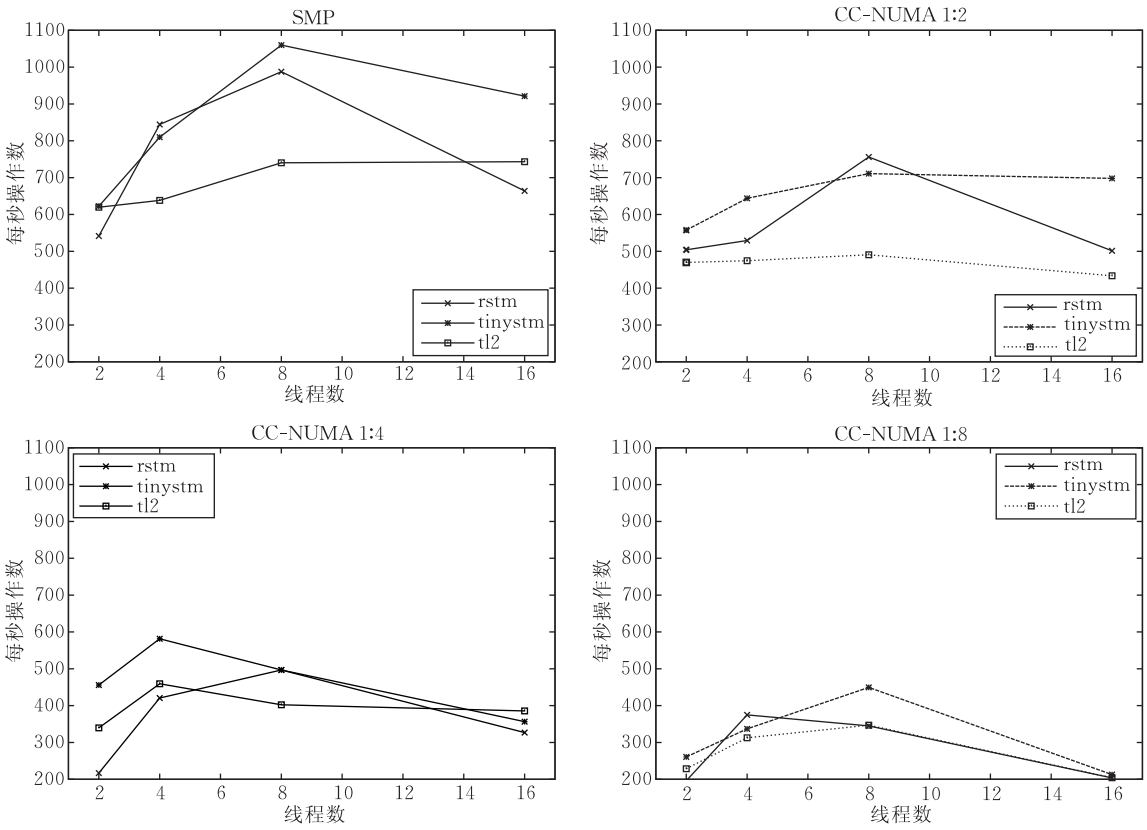


图 3　在 SMP 和 CC-NUMA 机器上使用 RSTM/TinySTM/TL2 测试 Stmbench7 的结果

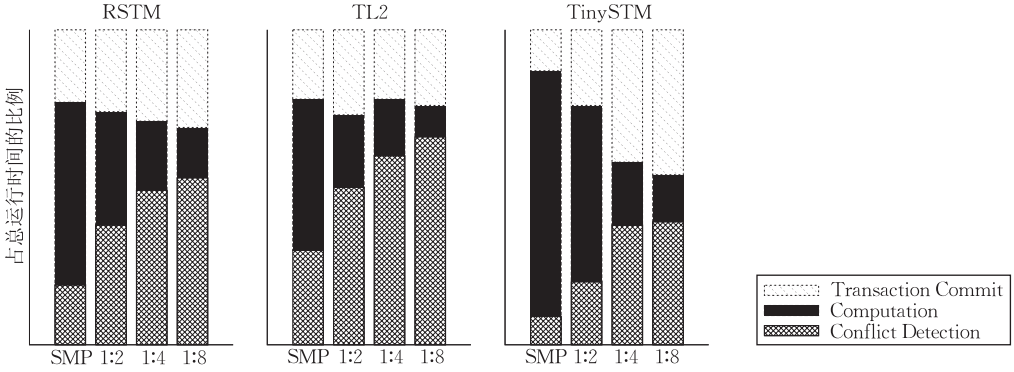


图 4　在 SMP 和 CC-NUMA 系统上测试 Stmbench7 的时间分布(线程数 16)

务提交上,在 1∶2 时这些操作也占用了 65%左右的时间,大大高于 SMP 机器上的 40%左右. 在 SMP 机器上运行良好的事务内存系统,显然无法适应 NUMA 结构,而这主要的原因是事务的失败率提高了. 这里给出 RSTM 在线程数 16 时的运行时信息,详细数据如表 1 所示.

**表 1　RSTM 运行 Stmbench7 的事务数量统计信息(线程数 16)**

| | Total | Abort | Abort/% |
|---|---|---|---|
| SMP | 195102282 | 6585081 | 3.38 |
| NUMA 1∶2 | 118070019 | 37755741 | 31.98 |
| NUMA 1∶4 | 87767680 | 30138633 | 34.34 |
| NUMA 1∶8 | 69682022 | 27364130 | 39.27 |

其中 Total 表示系统运行过程中启动的事务总量,Abort 表示失败的事务数量,可以看出,在 NUMA 机器上运行的事务失败率是 SMP 机器的 10 倍左右,这正好击中了事务内存的软肋:如果事务之间冲突较多,那么系统将耗费大量时间来进行事务的回滚、重启,事务重启之后,所有的冲突检测都需要重新进行,这都将占用系统大量的计算资源. 结论很简单,事务内存并不适用于事务之间冲突频繁的情形.

因此,本来在 SMP 机器上运行良好的事务内存应用,照搬到 CC-NUMA 机器上运行就不一定适用了. 那么,如何在 CC-NUMA 机器上降低事务的失败率,成为提高系统性能亟待解决的问题.

## 4 NUMA 敏感的冲突规避方法 PBC

从图 1 可以看出,传统事务内存的设计思路是,尽量提高系统中同时运行的事务数量,以增加并发度来提高性能,相对于锁的"悲观"并行来说,这是一种"乐观"的并行执行方式. 两个事务一旦发生数据访问冲突,则由"竞争管理算法"选择其中一个重启. 这里隐藏了事务内存的一个重要假设,就是大多数事务都可以一次性成功提交. 如果事务之间的冲突非常频繁,那么系统将耗费大量计算资源进行事务操作,从上一节的实验结果可以看出,这对系统性能的影响是巨大的.

由于 CC-NUMA 结构特性的影响,事务冲突的数量和比例大大增加,系统在冲突检测等事务相关操作上耗费了大量计算资源,影响了系统性能. 那么,如何减少事务冲突呢? 最极端的办法就是采用类似于锁的"悲观"并行手段,限制系统中同时只有一个事务运行,不过此时就丧失了系统的并行性. 这里给出一种权衡的方法,整体上仍然保持事务内存的"乐观"并行,但在可能发生冲突的事务上采取"悲观"并行策略,这需要在事务开始之前进行一次调度,称为冲突规避.

冲突规避在事务启动之前进行,根据系统当前情况作出判断,如果该事务发生冲突的可能性较大,则暂时挂起事务,否则让事务开始执行,流程如图 5 所示.

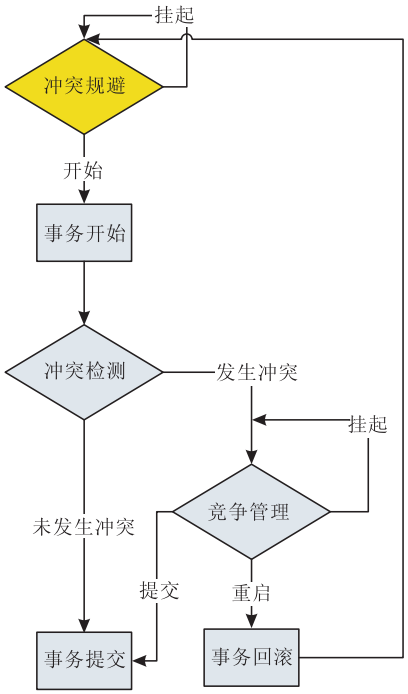


图 5 使用了冲突规避的事务执行流程

需要注意的是,传统的事务处理流程中,只有冲突的"事后处理",而冲突规避则可以视为冲突的"事前预防". 结合 NUMA 结构的特性,这里给出一种 NUMA 敏感的冲突规避算法 PBC(Predict Before Conflict),同时考虑事务之间的历史冲突情况和 NUMA 结构特性来预测事务的冲突可能性. 首先,将每个事务赋予唯一的事务 ID. 定义 $D(a,b)$ 为两个事务的"距离",如果事务 $a,b$ 在同一个处理器上运行(在不同的核心上),则"距离"为 0,否则为两个事务所在处理器之间的互联距离,$D$ 代表了两个事务发生冲突后的操作代价系数. 事务内存系统在运行时记录事务之间的冲突情况,系统记录事务 $a,b$ 之间的冲突次数为 $P(a,b)$,且 $P(a,b)=P(b,a)$. 在事务 $a$ 开始之前,系统根据当前正在运行的事务集合 $Q$,计算冲突可能

$$S_a=\sum P(a,i)\times D(a,i)\mid \forall i\in Q$$

设定阈值 $Y$,当 $S_a\geq Y$ 时,挂起事务 $a$ 直到 $S_a<Y$. 算法流程如图 6 所示.

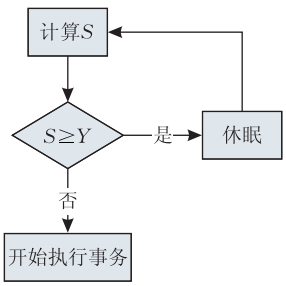


图 6 PBC 算法流程图

设事务总数为 $n$,下面给出事务启动时的冲突规避伪代码:

```
beginTransaction(ID) {
  S=0
  do {
    if (S>0) {sleep; S=0;}
    for (i=0; i<n; i++)
        if (i in Q) S+=P[ID,i] * D(ID,i);
    //计算冲突可能 S
  } while (S>=Y);
}
```

算法的时间复杂度是 $O(n)$,需要系统维护一个二维数组 $P$,规模为 $n^2$,实验时发现,$P$ 是一个稀疏矩阵,并不会占用太多的存储空间. 实际上,算法的核心问题是阈值 $Y$ 的确定,在实验中主要使用了经验数值. 根据实验结果发现,如果 $Y$ 值过大,则冲突规避的作用不明显,反之 $Y$ 值过小,则等待事务偏多,影响了系统的并行性. 为了更好地适应具体的测试程序,这里给出一种自适应的阈值调节方法,原则

有 3 点：(1)$Y$ 的初始值应该偏小；(2)随着冲突的减少，$Y$ 值应该逐渐增加；(3)随着冲突的增加，$Y$ 值应该逐渐减小到 $S$.

在实现过程中，由各个线程维护自己的阈值，也就是说，系统中并不存在一个全局的 $Y$ 值. $Y$ 的初始值设为系统中处理器的最大互联距离，例如全互联系统中 $Y$ 的初始值为 1. 事务成功提交时，$Y=Y+1$；事务发生冲突时，$Y=\frac{Y+S}{2}$.

## 5　实验结果和分析

实验环境与第 3 节相同，冲突规避方法基于 RSTM 实现，命名为 RSTM-$p$. 实验结果如图 7 所示.

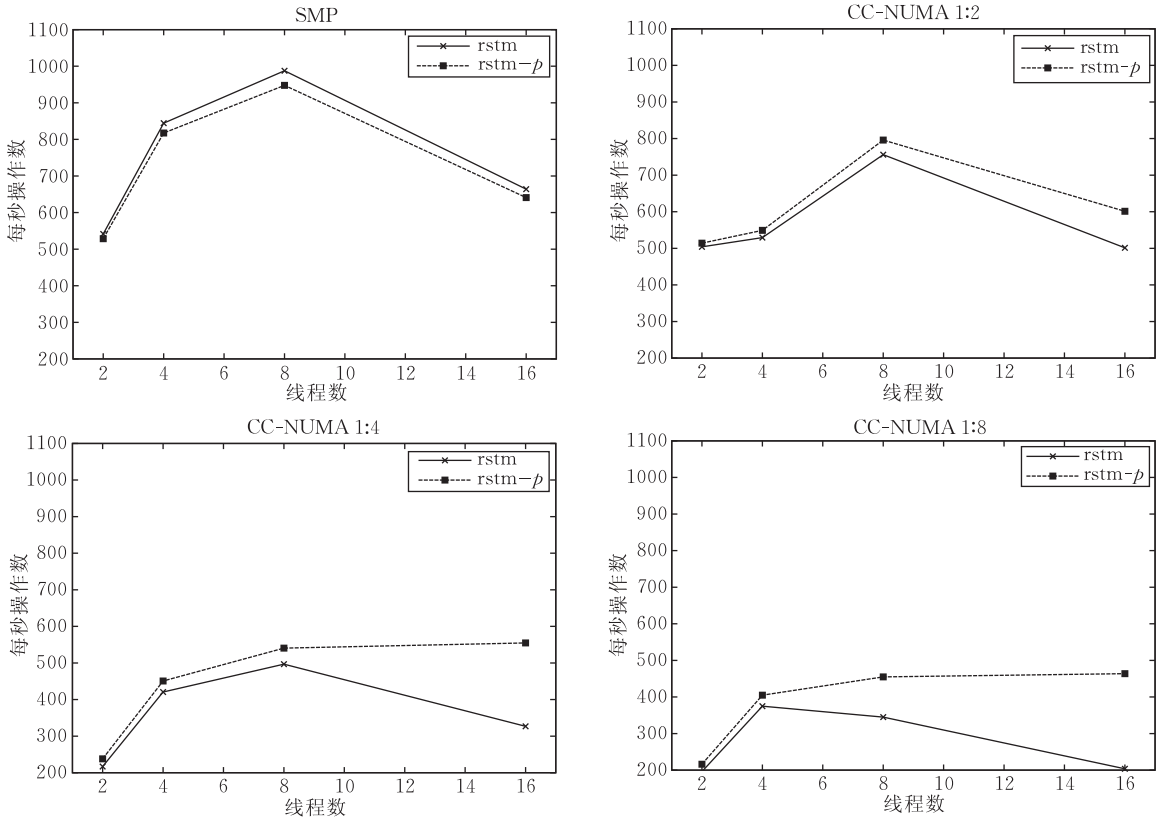


图 7　在 SMP 和 CC-NUMA 机器上使用 RSTM/RSTM-$p$ 测试 Stmbench7 的结果

此时，在 SMP 机器上，冲突规避 PBC 并没有带来性能的提高，反而使性能略有下降，这主要是因为冲突规避 PBC 算法本身有一定的计算开销所致. 在 CC-NUMA 机器上，PBC 方法开始发挥出优势，当本地、远程访存延迟比为 1∶2 时，RSTM-$p$ 的性能有了一定的提高，尤其是在线程数 16 的情况下，性能提高了 20%左右；随着远程访存延迟的增加，在 NUMA 1∶4 的情况下，RSTM-$p$ 在线程数 16 时的性能表现，略高于线程数 8 时的成绩，呈现出了可扩展性，这完全超越了之前 RSTM 的表现；在 NUMA 1∶8 时，PBC 的优势进一步凸显，在线程数 16 时的性能已经超过了之前 RSTM 的一倍多.

为了了解冲突规避算法 PBC 的计算开销，系统对 3 个事务内存系统在线程数 16 时的运行时信息，包括冲突检测时间、事务提交时间、事务处理之外的正常计算代码运行时间，尤其是冲突规避过程的 PBC 预测时间进行了采集，将这些数据规范化之后的结果如图 8 所示. 图 8 中“Conflict Preventing”表示冲突规避过程占用的时间，“Transaction Commit”表示事务提交占用的时间，“Conflict Detection”表示事务之间冲突检测占用的时间，“Computation”表示事务处理之外的正常计算代码执行时间.

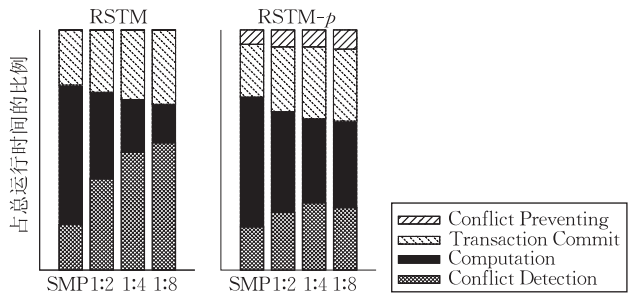


图 8　在 SMP 和 CC-NUMA 机器上测试 Stmbench7 的时间分布(线程数 16)

可以看出,在冲突规避 PBC 的介入下,系统在 CC-NUMA 机器上运行的"Computation"时间比例有了明显升高,虽然冲突规避占用了一部分计算资源,但是带来的好处也是很明显的.而且,在 CC-NUMA 机器上相应的事务处理时间比例有了一定的下降.而随着远程内存访问延迟的增加,冲突规避 PBC 过程占用的时间比例上升并不明显,基本上稳定在 10%左右,说明 PBC 算法并不依赖于远程访存操作,适用于 CC-NUMA 机器.

为了比较事务冲突失败的数量信息,采集 RSTM-$p$ 执行过程中的事务数量信息(线程数 16)如表 2 所示.

**表 2 RSTM-$p$ 运行 Stmbench7 的事务数量统计信息**

| | Total | Abort | Abort/% |
|---|---|---|---|
| SMP | 189312238 | 6365178 | 3.36 |
| NUMA 1∶2 | 109989932 | 20040165 | 18.22 |
| NUMA 1∶4 | 89732168 | 18027192 | 20.09 |
| NUMA 1∶8 | 70232742 | 16104367 | 22.93 |

可以明显看出,在 CC-NUMA 机器上使用冲突规避 PBC 算法来运行的事务失败率降低为原来 RSTM 的一半左右,PBC 方法有效地降低了事务失败率.

## 6 结 论

现有的事务内存研究主要面向多核处理器和 SMP 机器,并没有考虑到 NUMA 结构的影响.本文通过实验证明,NUMA 特性带来的本地与远程访存延迟差异对事务内存的性能有一定的影响.

本文提出了 NUMA 敏感的冲突规避方法 PBC,在事务启动之前通过冲突预测算法来估计该事务发生冲突的可能性,并以此为依据对事务进行调度,可以有效地减少事务冲突,使事务内存在 CC-NUMA 机器上获得高性能.

NUMA 敏感的事务内存研究还有很多改进和扩展的空间,下一步工作中将测试现有的竞争管理策略与冲突规避方法的适配情况.总之,事务内存是多核时代很有意义的研究工作,在当前普遍使用多核处理器和 CC-NUMA 体系结构搭建高性能服务器的情况下,如何让事务内存在 CC-NUMA 结构上高效运行,是学术界应当关注的方向.

## 参 考 文 献


[1] Adve Sarita V, Gharachorloo Kourosh. Shared memory consistency models: A tutorial. IEEE Computer, 1996, 29(12): 66-76

[2] William N Scherer, III. Synchronization and concurrency in user-level software systems[Ph. D. dissertation]. Rochester, NY, USA: University of Rochester, 2006

[3] Greenwald Michael. Non-blocking synchronization and system design[Ph. D. dissertation]. Stanford University, Also appears as Stanford University Technical Report STAN-CS-TR-99-1624, Stanford University, Stanford, CA, 1999

[4] Herlihy Maurice. The art of multiprocessor programming// Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing. Denver, Colorado, USA, 2006: 1-2

[5] Kongetira P, Aingaran K, Olukotun K. Niagara: A 32-way multithreaded Sparc processor. IEEE Micro, 2005, 25(2): 21-29

[6] Meadows Larry. Openmp 3.0—A preview of the upcoming standard//Proceedings of the 3rd International Conference on High Performance Computing and Communications. Berlin, Heidelberg, 2007: 4

[7] Smaragdakis Yannis, Kay Anthony, Behrends Reimer, Young Michal. General and e_cient locking without blocking//Proceedings of the 2008 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness. Seattle, Washington, 2008: 1-5

[8] Herlihy Maurice. The transactional manifesto: Software engineering and non-blocking synchronization//Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation. Chicago, IL, USA, 2005: 280

[9] Larus James R, Rajwar Ravi. Transactional Memory. San Rafael, CA: Morgan & Claypool, 2006

[10] McDonald Austen, Carlstrom Brian D, Chung JaeWoong, Minh Chi Cao, Chafi Hassan, Kozyrakis Christos, Olukotun Kunle. Transactional memory: The hardware-software interface. IEEE Micro, 2007, 27(1): 67-76

[11] Harris Tim, Cristal Adrian, Unsal Osman S, Ayguade Eduard, Gagliardi Fabrizio, Smith Burton, Valero Mateo. Transactional memory: An overview. IEEE Micro, 2007, 27(3): 8-29

[12] Date C J. Introduction to Database Systems. Reading, MA: Addison Wesley/Pearson, 2006

[13] Carlstrom Brian David. Programming with transactional memory[Ph. D. dissertation]. Stanford University, Stanford, CA, 2008

[14] Drepper Ulrich. Parallel programming with transactional memory. Queue, 2008, 6(5): 38-45

[15] Manassiev Kaloian, Mihailescu Madalin, Amza Cristiana. Exploiting distributed version concurrency in a transactional memory cluster//Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York, NY, USA, 2006: 198-208

[16] Amza Cristiana, Cox Alan L, Dwarkadas Sandhya, Keleher Pete, Lu Honghui, Rajamony Ramakrishnan, Yu Weimin, Zwaenepoel Willy. Treadmarks: Shared memory computing on networks of workstations. Computer, 1996, 29(2): 18-28

[17] Bocchino Robert L, Adve Vikram S, Chamberlain Bradford L. Software transactional memory for large scale clusters//

Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York, NY, USA, 2008: 247-258

[18] Kotselidis Christos, Ansari Mohammad, Jarvis Kim, Luján Mikel, Kirkham Chris, Watson Ian. DiSTM: A software transactional memory framework for clusters//Proceedings of the 2008 37th International Conference on Parallel Processing. Washington, DC, USA, 2008: 51-58

[19] Bonachea Dan. Gasnet specification, v1. 1. Technical Report, Berkeley, CA, USA, 2002

[20] Laurent Baduel, Françoise Baude, Denis Caromel, Arnaud Contes, Fabrice Huet, Matthieu Morel, Romain Quilici. Grid Computing: Software environments and tools. Chapter Programming, Deploying, Composing, for the Grid. Springer-Verlag, January 2006

[21] Wang Ruibo, Lu Kai, Lu Xicheng. Investigating transactional memory performance on ccnuma machines//Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing. New York, NY, USA, 2009: 67-68

[22] Wang Ruibo. Investigating software transactional memory on big smp machines//Proceedings of the ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing. Daegu, Korea, 2009: 507-509

[23] Martin Milo M K, Sorin Daniel J, Beckmann Bradford M, Marty Michael R, Xu Min, Alameldeen Alaa R, Moore Kevin E, Hill Mark D, Wood David A. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. SIGARCH Computer Architecture News, 2005, 33(4): 92-99

[24] Guerraoui Rachid, Kapalka Michal, Vitek Jan. Stmbench7: A benchmark for software transactional memory//Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007. Lisbon, Portugal, 2007: 315-324

[25] Marathe Virendra J, Spear Michael F, Heriot Christopher, Acharya Athul, Eisenstat David, William N Scherer III, Scott Michael L. Lowering the overhead of nonblocking software transactional memory//Proceedings of the 1st ACM SIGPLAN Workshop on Languages, Compilers, and Hardware Support for Transactional Computing. Ottawa, Canada, 2006

[26] Felber Pascal, Fetzer Christof, Riegel Torvald. Dynamic performance tuning of word-based software transactional memory//Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York, NY, USA, 2008: 237-246

[27] Dice Dave, Shalev Ori, Shavit Nir. Transactional locking II//Proceedings of the 20th International Symposium on Distributed Computing. Stockholm, Sweden, 2006

**WANG Rui-Bo**, born in 1981, Ph. D. candidate. His research interests include operating system and high performance computing.

**LU Xi-Cheng**, born in 1946, professor, Ph. D. supervisor, member of Chinese Academy of Engineering. His research interests include distributed computing, computer networks and communications.

**LU Kai**, born in 1973, professor, Ph. D. supervisor. His research interests include operating system and high performance computing.

**WANG Shao-Gang**, born in 1979, Ph. D.. His research interests include parallel computer architecture and high performance storage system.

**Background**

This research is supported by the National Natural Science Foundation under grant No. 60621003. With the development of multi-core and many-core processors, desktop developers started using parallel programs to exploit processor cores. Most parallel programs used locks as synchronization mechanism, but it was not easy to avoid deadlock, priority inversion, lock convoying, etc. Given the above problems, it was not surprising to see a large body of research aimed at making parallel programming easier. One of the most appealing solutions was Transactional Memory (TM). TM is a synchronization primitive provides atomic and isolated execution for regions of code. Developers could simply use the atomic primitive to define a critical section, and the data race inside the critical section would be resolved automatically. TM was originally designed for shared memory system such as SMP systems. With the development of Direct Connect Architecture, the processor was optionally directly connected to other CPUs through a proprietary extension running on top of additional natively implemented HyperTransport interface allowing support of a cache-coherent NUMA (CC-NUMA) multi-CPU memory access protocol. In other words, the modern SMP systems were CC-NUMA now. Under NUMA, a processor could access its own local memory faster than non-local memory, which was totally different from SMP. The performance of existing TM implementations running on CC-NUMA system was poor, slow non-local memory access latency might be an issue. To solve the problem and reduce the transaction conflicts, the conflict preventing method was presented. A new scheduler was added before each transaction's starting. The scheduler was using a forecasting algorithm to predict the transaction's future conflict probability. The probability was used to schedule the transaction, suspend or start. With the conflict preventing method, the performance of TM was improved on NUMA system. The authors appreciate that many helpful suggestions and discussions from Ian Rogers with Azul Systems.