事务内存专题讲座01

什么是事务内存？

张坤龙

zhangkl@tju.edu.cn

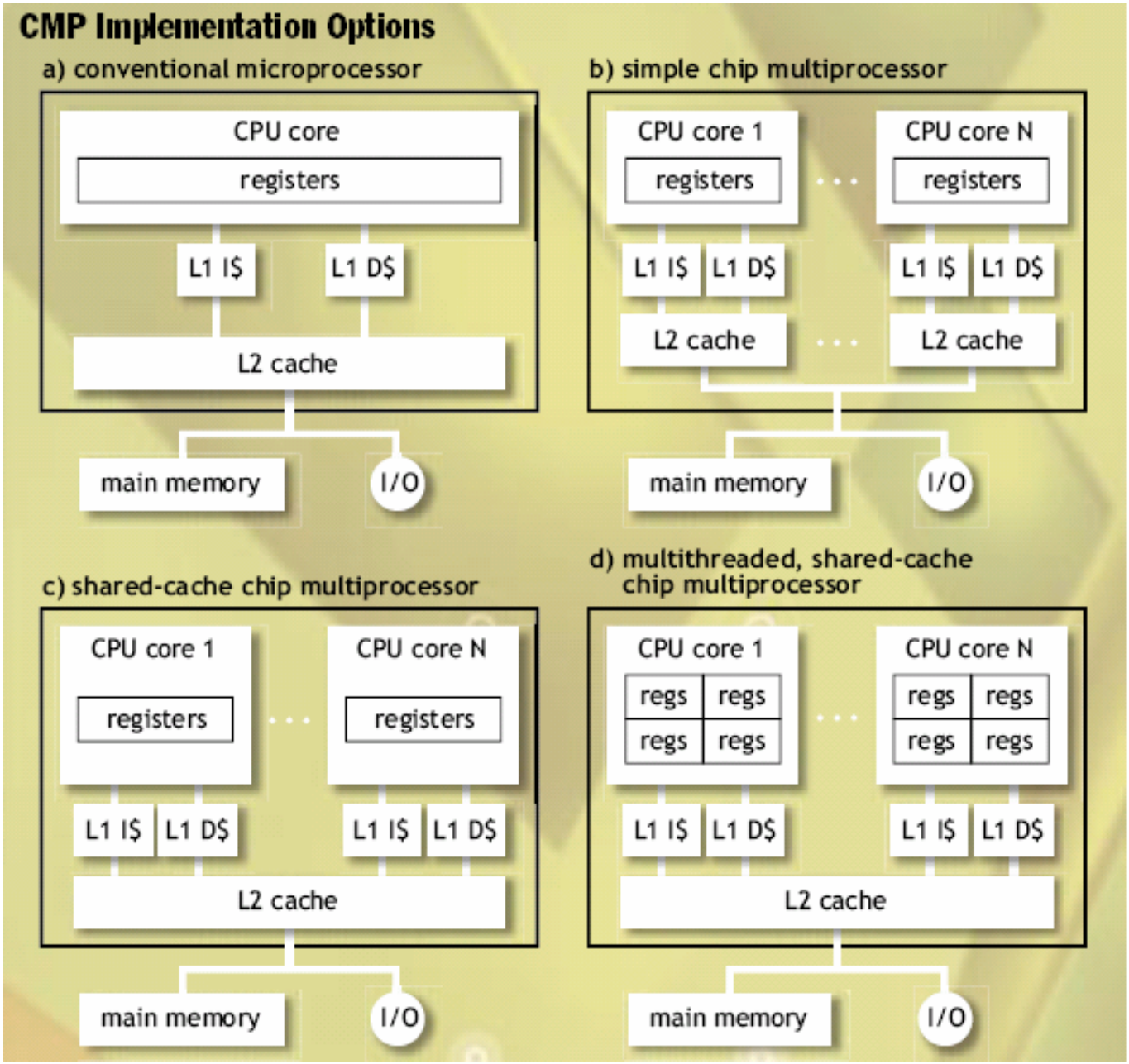
2007-04-02第1版，2007-06-29第2版

目录

- 多核处理器
- 并行程序设计
- 数据库技术
- 事务内存技术

多核处理器

- 由于能耗以及散热方面的原因，现在已经不再通过提高时钟频率来提升处理器的性能。
- 但是，摩尔定律（芯片上的晶体管数目大约每过两年就翻一番）在现在和将来的一段时间内仍然有效。
- 因此，在单个芯片上集成多个CPU内核来构造多核处理器（chip multiprocessors, multicore processor）。



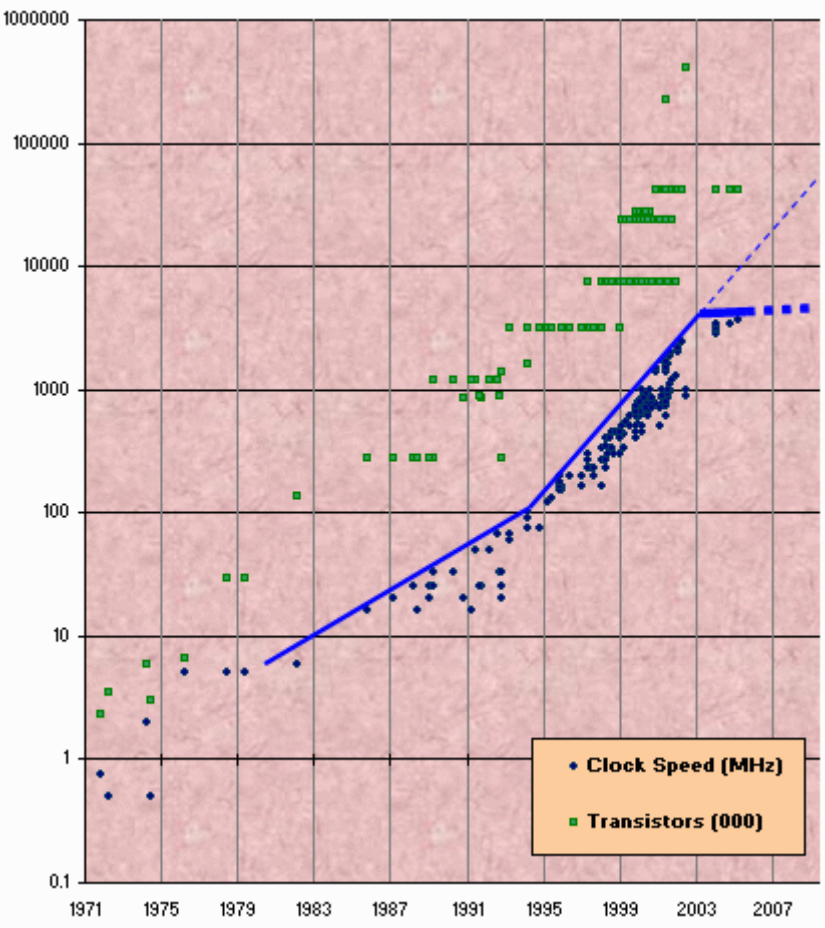
•左图给出了多核处理器的几种实现方式[1]。

•2005年春季，AMD公司和Intel公司发布了各自的首款双核处理器。

•2006年9月，Intel公司对外展示了80核处理器的原型 。

•在多核处理器时代，单个CPU内核的性能提升程度有限，甚至还有可能不如当前的单核处理器。

免费午餐结束了！



- 在单核处理器时代，顺序程序无需作任何修改就能享受到处理器性能提高带来的好处。
- 而在多核处理器时代，只有并行程序才有可能充分利用已有的多个CPU内核，并且在以后无需修改就能享受到CPU内核数目增加带来的好处。
- Herb Sutter: “The biggest sea change in software development since the OO revolution is knocking at the door, and its name is Concurrency.”[2]

目录

- 多核处理器
- 并行程序设计
- 数据库技术
- 事务内存技术

并行程序设计现状

- 并行程序设计只与少数人相关，不像顺序程序设计那样广为人知。造成这种现状的原因有：“免费午餐”、并行程序设计比顺序程序设计困难。

- 心理学研究表明，人把注意力发散开来同时处理多个任务的能力是有限的。

- 分析并行程序比分析顺序程序困难。在顺序程序分析中只需要考虑上下文，而在并行程序分析中还需要考虑同步。已经证明，即使只有两个线程，将上下文和同步结合起来分析也是一个不可判定问题[3]。

- 在程序设计模型、程序设计语言、程序设计工具等方面，并行程序设计落后于顺序程序设计

并行程序设计模型

- 数据并行性（data parallelism)
  - 各处理器在不同的数据上执行同一个任务
  - 需要进行数据划分
  - 适合于数值计算，不通用

- 任务并行性（task parallelism)
  - 各处理器在相同的或者不同的数据上执行不同的任务
  - 需要进行任务划分
  - 显式的同步，使得编程困难

基于锁的同步

- 粗粒度的锁虽然易于使用，但是性能低，因为它导致对锁的竞争、使得无关的操作顺序执行。
- 细粒度的锁虽然性能高，但是难于使用，例如双端队列的并行版本用细粒度锁就难于实现。
- 存在优先级反转（priority inversion）、护航（convoy ）、死锁等的问题。
- 可组合性差，例如从一个链表移动一个元素到另一个链表时，必须使用低性能的粗粒度锁来避免可能发生的死锁。

新的同步机制?

- 不使用锁：免锁算法（lock-free algorithms)，免锁数据结构（lock-free data structures)
- 使用新的同步原语，例如CAS（Compare And Swap）
- 免锁数据结构很难设计，例如免锁链表[4,5]

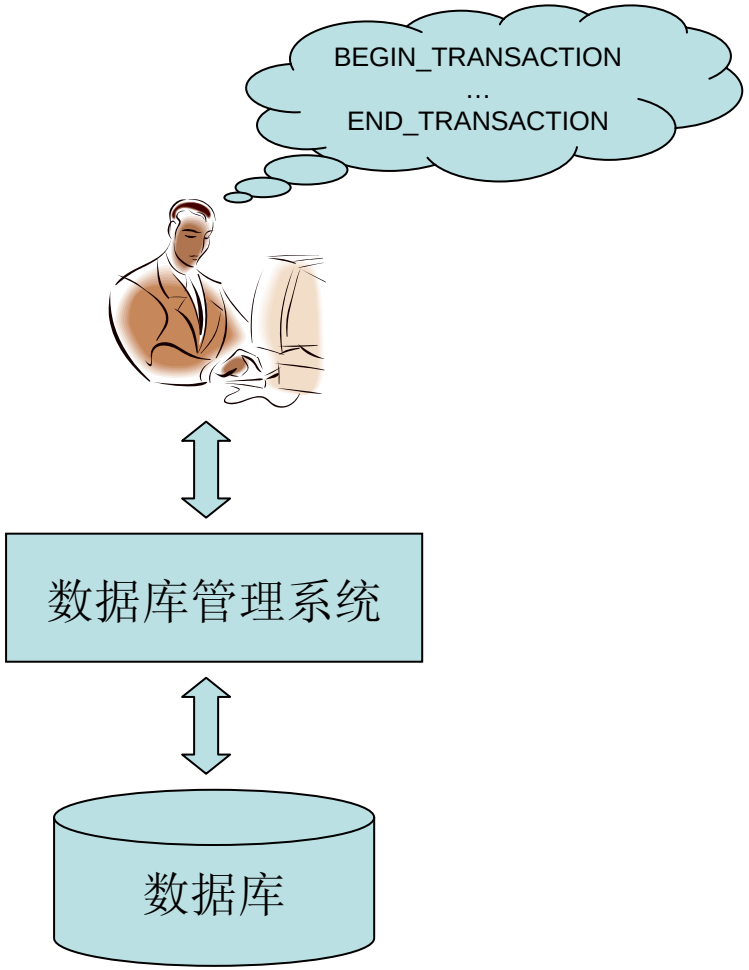
```
int CAS(int *addr,int old,int new)
{
        atomic {
                if  (*a==old) {
                        *a=new;
                        return true;
                } else {
                        return false;
                }
        }
}
```

目录

- 多核处理器
- 并行程序设计
- 数据库技术
- 事务内存技术

数据库

- 数据库是数据的集合，通常存放在磁盘上，应用于数据管理
- 只能通过事务来访问数据库（数据库事务）
- 数据库管理系统负责正确执行事务
- 程序员只需将对数据库的操作封装为事务，无需考虑复杂的事务同步问题



事务

- 事务是用户定义的一个数据库操作序列，具有以下特性：
  - 原子性 (Atomicity)：事务的诸操作要么都做（事务提交），要么都不做（事务夭折）
  - 一致性 (Consistency)：事务执行的结果必须是使数据库从一个一致状态变成另一个一致状态
  - 隔离性 (Isolation)：一个事物的执行不能被其他事务干扰，即并发执行的各个事务之间不能互相干扰
  - 持久性 (Durability)：一个事务一旦提交，对数据库中的数据的改变是永久性的

数据库管理系统

- 数据库管理系统只提交不和其他事务发生冲突的事务，当数据库管理系统发现两个事务冲突时，它会自动选择其中一个夭折重做

- 当两个未提交的事务对同一个数据进行访问时，只要其中有一个写操作，那么这两个事务之间就会发生冲突

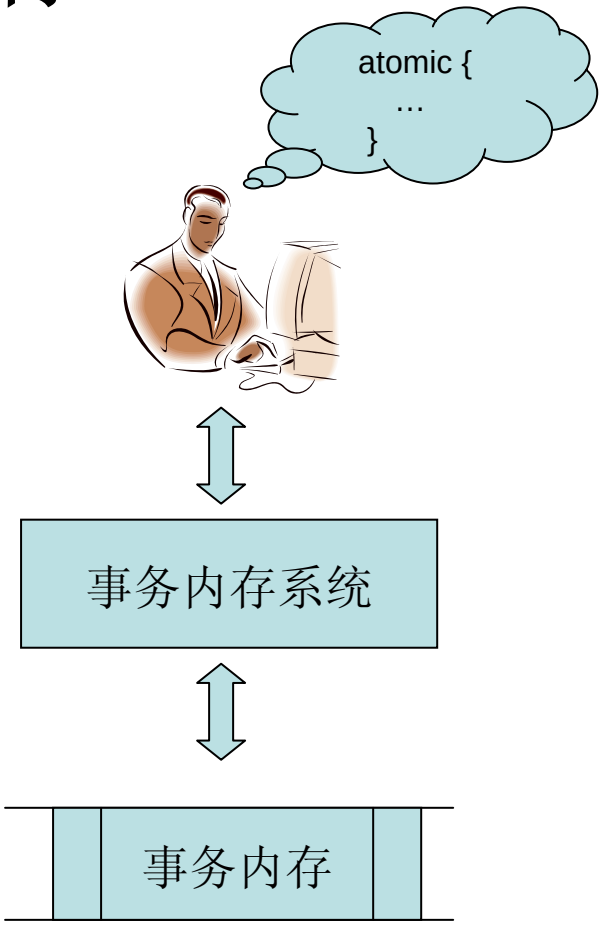
- 冲突造成的数据不一致性包括三类
  - 丢失更新：一个事务写过的数据被另外一个事务写
  - 读脏数据：一个事务写过的数据被另外一个事务读
  - 不可重复读：一个事务读过的数据被另外一个事务写

目录

- 多核处理器
- 并行程序设计
- 数据库技术
- 事务内存技术

事务内存

- 事务内存是数据的集合，通常存放在主存中，应用于程序设计
- 可以通过事务来访问事务内存（内存事务）
- 事务内存系统负责正确执行事务
- 程序员只需将对事务内存的操作封装为事务，无需考虑复杂的事务同步问题



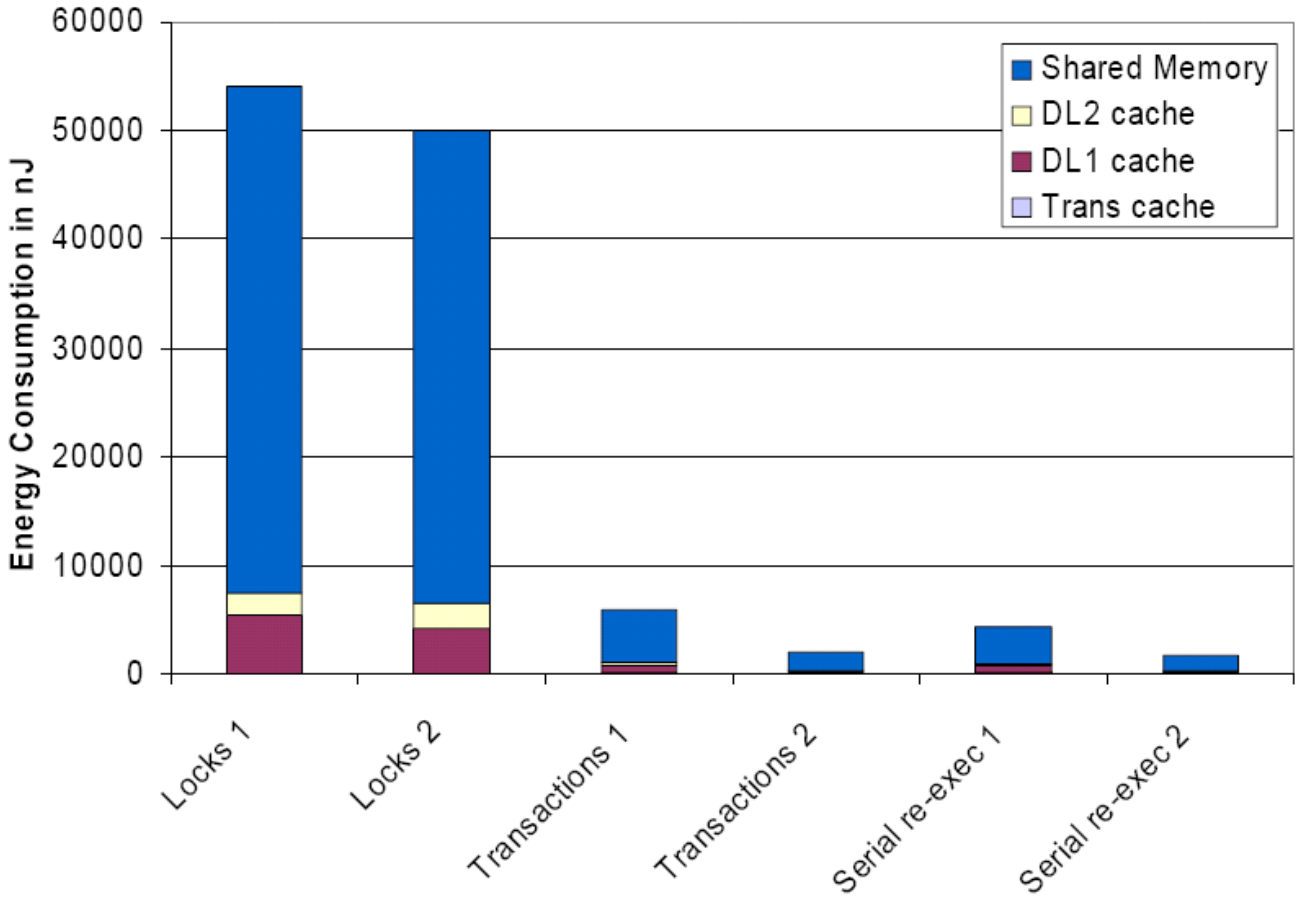
事务内存和数据库

- 数据库中的数据存储在磁盘上，事务内存中的数据存储在主存中。
- 数据库中的数据需要持久存储，而事务内存中的数据不需要持久存储。
- 所有对磁盘数据的访问都必须通过数据库管理系统来进行。但是对主存数据的访问不太可能只是来自于事务内存系统，事务内存系统必须和现有的程序设计语言、程序库、操作系统等共存。
- 数据库管理系统通常是一个大而复杂的软件系统。事务内存系统则相对简单，可以完全用软件实现（软件事务内存系统），或者完全用硬件实现（硬件事务内存系统），或者用混合方式实现（混合事务内存系统）。

事务内存的优点

- 事务内存一方面像粗粒度锁一样易于使用，另一方面其性能又能与细粒度锁匹敌（甚至超过！）。
- 如果不使用锁来实现，事务内存就没有优先级反转、护航、死锁等问题（非阻塞事务内存）
- 通过支持嵌套事务，事务内存具有较好的可组合性
- 与所相比较，有利于在出错后恢复原来的状态
- 与锁相比较，事务内存可以节省能源[10]

节能

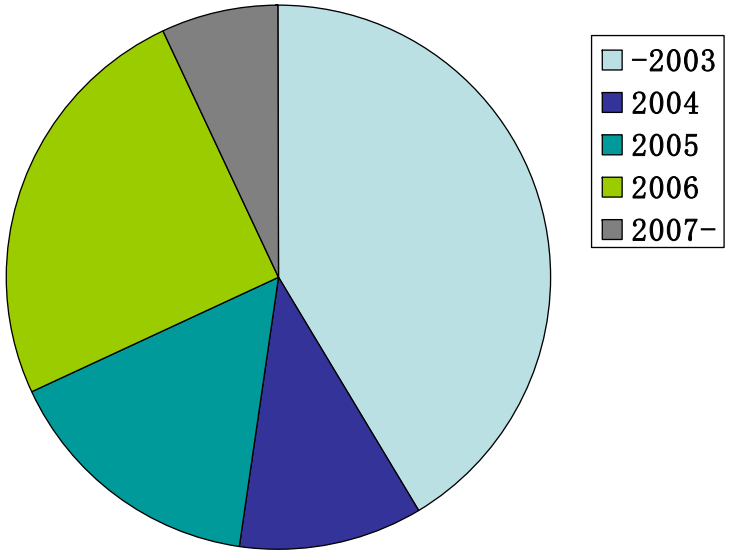

60000
50000
40000
30000
20000
10000
0
Energy Consumption in nJ
Shared Memory
DL2 cache
DL1 cache
Trans cache
Locks 1
Locks 2
Transactions 1
Transactions 2
Serial re-exec 1
Serial re-exec 2

事务内存的研究状况

- 1977年，Lomet提出了在程序设计语言中引入类似数据库事务的抽象的想法，但未给出实现[6]
- 1993年，Herlihy和Moss给出了一个硬件事务内存系统[7]
- 1995年， Shavit和Touitou给出了一个软件事务内存系统[8]
- 最近几年，事务内存成为研究热点，出现了一些（软件，硬件、混合）事务内存系统[9]

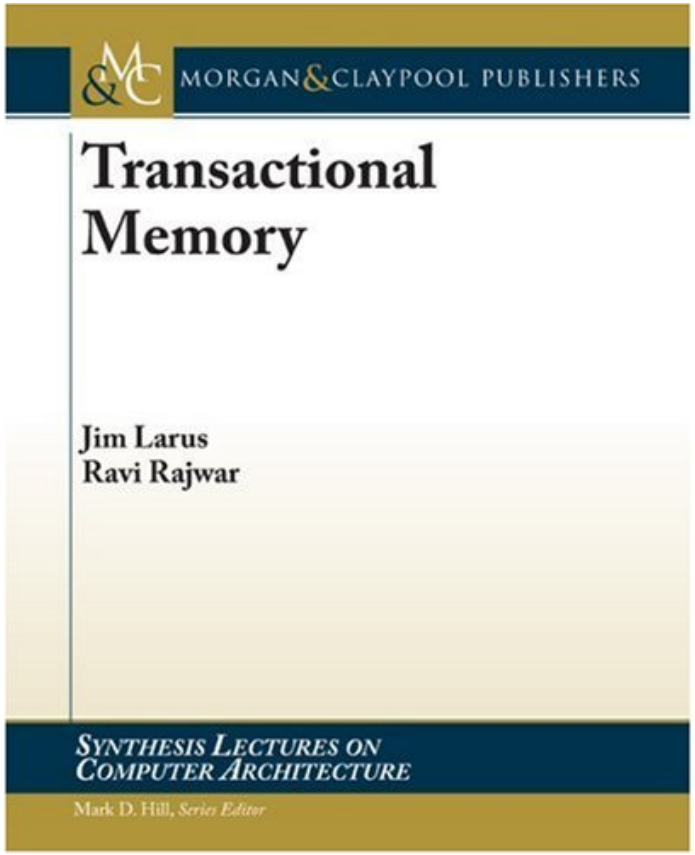
对参考文献的统计

- Transactional Memory Online Bibliography网站 http://www.cs.wisc.edu/trans-memory/biblio/



- 2007年5月31统计，共159篇参考文献

《Transactional Memory》



Title:
Transactional Memory
Authors:
James Larus, Ravi Rajwar
Publisher:
Morgan & Claypool Publishers
Paperback:
221 pages
Publish date:
January 12, 2007
Language:
English
ISBN:
1598291246

参考文献

1. K.Olukotun and L. Hammond. **The Future of Microprocessors**. ACM Queue, 3(7):26–29, 2005.
2. Herb Sutter. **The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software**. Dr. Dobb's Journal. Vol 30. No 3. March 2005.
3. G. Ramalingam. **Context-sensitive synchronization-sensitive analysis is undecidable**. ACM TOPLS 22(2):416–430, 2000.
4. J. Valois. **Lock Free Linked Lists Using Compare-and-Swap**. In Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing, pages 214-222, August 1995.
5. T. Harris. **A Pragmatic Implementation of Non-Blocking Linked Lists**. In Proceedings of the 15th International Symposium on Distributed Computing, pages 300-314, October 2001.

参考文献（续）

6. D.B. Lomet. **Process structuring, synchronization, and recovery using atomic actions**. In Proceedings of ACM Conferences on Language Design for Reliable Software, 1977.
7. M. Herlihy and J. Moss. **Transactional memory: architectural support for lock-free data structures**. In Proceedings of ISCA, 1993.
8. Nir Shavit and Dan Touitou. **Software Transactional Memory**. In Proceedings of PODC, 1995.
9. J. Larus and R. Rajwar. **Transactional Memory**. Morgan & Claypool Publishers. 2007.
10. Tali Moreshet, R. Iris Bahar and Maurice Herlihy. **Energy-Aware Microprocessor Synchronization**: Transactional Memory vs. Locks. In Proceedings of Workshop on Memory Performance Issues, 2006.