C-DAC Four Days Technology Workshop

ON

Hybrid Computing – Coprocessors/Accelerators
Power-Aware Computing – Performance of Applications Kernels

hyPACK-2013
(Mode-1:Multi-Core)

Lecture Topic:

Transactional Memory - Multi-Core Processors

Venue : CMSD, UoHYD ; Date : October 15-18, 2013

Multi-Core Processors - Transactional Memory

Quick overview of what this Lecture is all about

- Background -Software Parallelization : a major issue for performance; Motivation
- Software Transactional Memory
- Transactional Memory Programming Paradigms
- Benchmarks Efforts TM on Multi-Core Processor Systems
- Conclusions

Multi-core Processors & Transactional Memory : Background

Source : Reference : [4], [6], [11],[12], [24],[25], [26], [36-63],

Multi-Core History

- Parallel computing has been used for HPCs and networking.
  - PRAM & other shared memory models - Limitations.
  - BSP & LogP (message passing models) were used.
    - Only for HPC specialists.
    - Demand complicated system analyze per application.
  - Clusters are becoming popular (NUMA, CC NUMA)

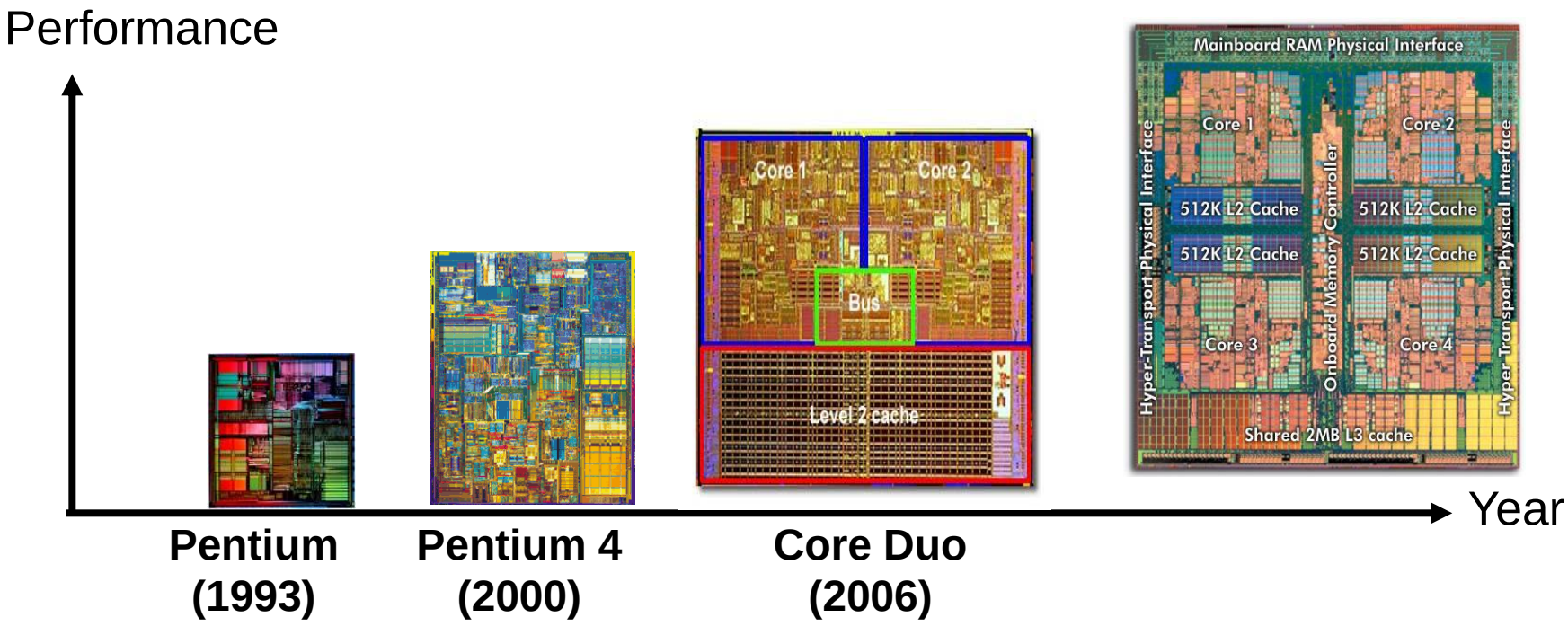
- HW (Scalability) Constraints force Multicore architectures.

- Today's parallel programming is just Mulit-Core Prog. & Mixed Prog. & Hybrid Computing

- Today's parallel programming based on *locks.*
  - *Coarse grained* code prevent parallelism, *fine grained* are hard to use.
  - Code reuse demands exposing *internal locks*.
  - No conventional way to connect *mutex* and its data.

Multi-Core Processors

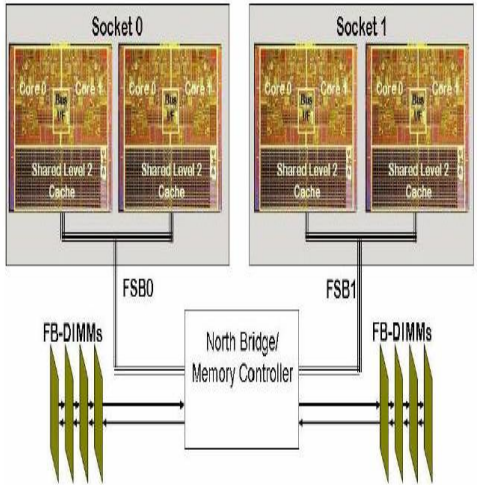
- No more frequency race
  - The era of multi cores
- Parallel programming is not easy
  - Split a sequential task into multiple sub tasks
  - Data Parallelism – Task Paralleism

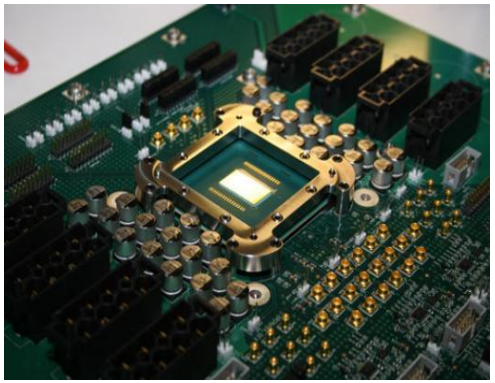


Multi-Cores - Parallel Programming -Difficulties

- Programmers Challenge :
- Identify independent pieces of a task that can be executed in parallel
  - Coordinate their Execution
  - Managing Communications
  - Managing Synchronization
- A program with a communication or synchronization bottleneck will be unable to take full advantage of the available Cores
- Scalable Programs that avoid such bottlenecks are surprisingly difficult to construct

Intel Quad Core (Clovertown)



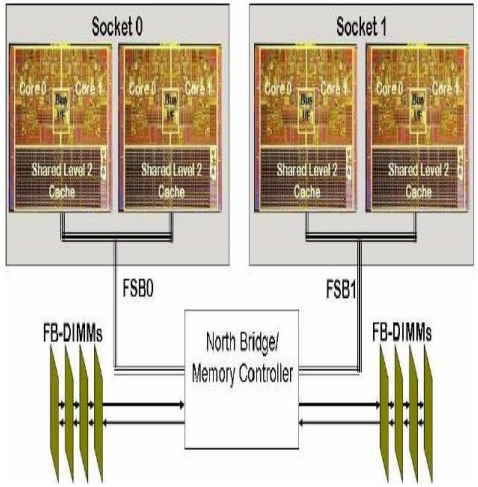


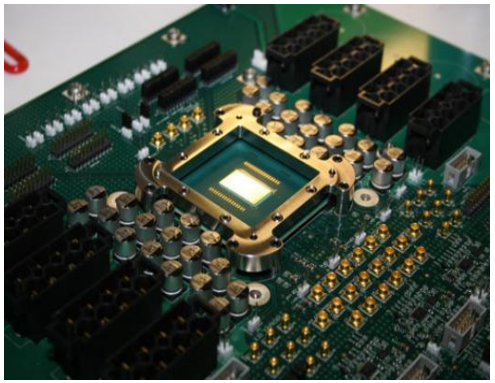
Intel's 80 core chip

Multi-Cores - Parallel Programming -Difficulties

- Challenge: taking advantage of Multi-Core
- Parallel Prog. is difficult with locks:
  - Deadlock, convoys, priority inversion
  - Conservative, poor composability
  - Lock ordering complicated
  - Performance-complexity tradeoff
- Transactional Memory in the OS
  - Benefits user programs
  - Simplifies programming

Intel Quad Core (Clovertown)





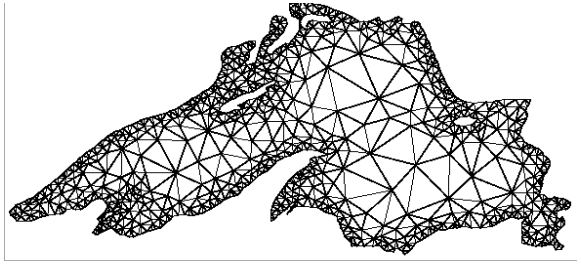
Intel's 80 core chip

Multi-Cores - Parallel Programming Difficulties

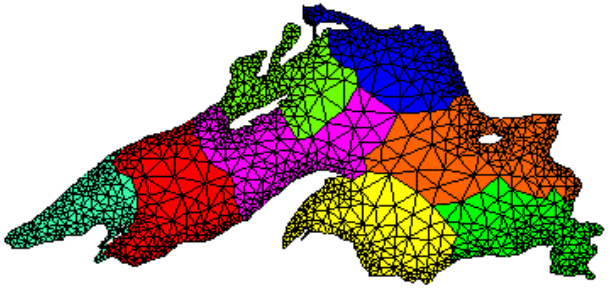
- Multicore architectures force us to rethink how we do synchronization
- Parallel programming has traditionally been considered using locks to synchronize concurrent access to shared data.
- Standard locking model won't work
- Lock-based synchronization, however, has known pitfalls: using locks for fine-grain synchronization and composing code that already uses locks are both difficult and prone to deadlock.
- Transactional model might
  - Software
  - Hardware
  - Programming Issues

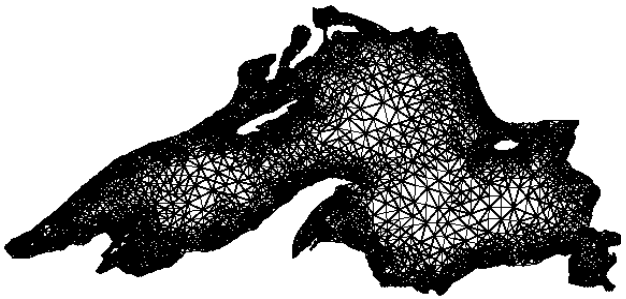
Multi-Cores - TM Challenges

- Commonly achieved via:
  - Threads for parallelism
  - Locks for synchronization



- Unfortunately, synchronization with locks is hard
  - **Option 1:** Coarse-grain locks
    - Simplicity
    - Decreased concurrency
  - **Option 2:** Fine-grain locks
    - Better performance maybe)
    - Increased complexity (bugs)
      - Deadlock, priority inversion, convoying, .

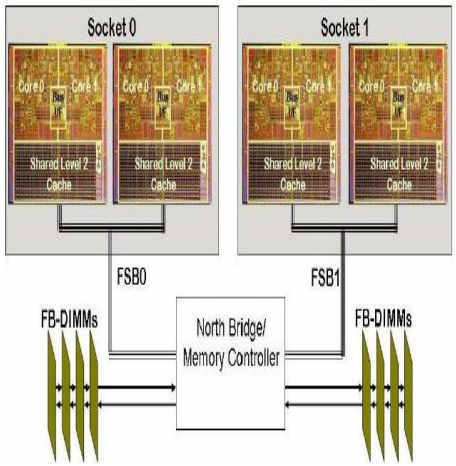




Multi-Cores - TM Challenges

- How to address Complex Interactions ?
  - Managing – Careful use of Locks
  - Critical Sections – Execute Automatically
  - Performance-complexity tradeoff
- Research in Transactional Memory
  - Apply the idea of transactions (popular in database systems) to computer memory
  - Proposed in Hardware and Software
  - Proposed extensions to Programming languages, implemented with a combination of compiler support and runtime libraries
  - Proposed Library interfaces
  - Propose Instruction set extensions

Intel Quad Core (Clovertown)



Intel's 80 core chip

Transactional Memory (TM)

- A Synchronization Mechanism to coordinate accesses to shared data by concurrent threads (An alternative to locks)

Transaction: A group of operations on shared data

An API Enhancement:

1. Abort in middle of a transaction
   - On encountering a error

```
Transaction {
    A = A – 10;
    B = B + 10;
    ...
    if (error)
        abort_transaction;
}
```

Source : Reference : [4], [6], [11],[12], [24],[25], [26], [36-63],

Transactional Memory (TM)

- What is a transaction?
  - Group of instructions in computer program:

```
atomic {
  if (x != NULL) x.foo();
  y = true;
}
```

  - Required properties: Atomicity, Isolation, Serializability

- Key idea: Use transactions to ease parallel programming
  - Locks → programmers define & implement synchronization
  - TM → programmers declares & system implements
    - Simple like coarse-grain locks & fast like fine-grain locks

Motivation – Transactional Memory ™

- Multi-core Processors are easily available
  - But writing parallel SW is hard

- Transactional Memory (TM) is a promising solution
  - Large atomic blocks simplify synchronization
  - Speed of fine-grain locks with simplicity of coarse-grain locks
  - But where are the benchmarks?

Motivation – Transactional Memory ™

- Threads + locking won't work
- Replace locking with a transactional API
  - New models; New languages; New systems
- Challenges to building TM systems
  - Common case behavior of parallel programs
  - TM virtualization
- Opportunities for systems beyond parallel programming
  - Multithreading for dynamic binary translation
  - Support for reliability, security, and fast memory snapshot

Source : Reference : [4], [6], [11],[12], [24],[25], [26], [36-63],

Intel C++ STM - Transactional Memory ™

- STM stands for Software Transactional Memory, a promising technology to help accelerate the creation of parallel applications.
- Transactional memory is proposed to simplify parallel programming by supporting "atomic" and "isolated" execution of user-specified tasks. It provides an alternate concurrency control mechanism that avoids these pitfalls and eases parallel programming.
- Intel® C++ STM Compiler Prototype edition 2.0 and runtime libraries for Intel transactional memory language construct extensions.
- The Transactional Memory C++ language constructs for parallel programming, understand the transaction memory programming model, and provide feedback on the usefulness of these extensions with Intel® C++ STM Compiler Prototype Edition.
- C and C++ programmers on Windows* and Linux*, using Intel's production compiler.

Source : Reference : [49], [50], [51-63],

Intel® C++ STM Compiler Prototype

- Great opportunity to explore Software Programming Models for parallel programming.
- What is new in Prototype Edition 3.0?
  - New language constructs **`__tm_atomic { …}`** else **`{ … }`** and **`__tm_wavier {…}`**
  - New function annotations **`__tm_safe`**, and **`tm_wrapping`** with support for registering commit and undo handlers for writing advanced transactional libraries
  - Transactional C++ new/delete, constructor and destructor support
  - TM annotation for template classes
  - Support for abort-on-exception semantics with explicit **`_tm_abort throw`** <exception>
  - New compiler and runtime optimizations
  - Support for transactional C++ STL library

Source : Reference : [49], [50], [51-63],

Intel® C++ STM Compiler Overview

Intel Transactional Memory compiler and Runtime Application Binary Interface

❖ Languages

❖ Library Design Principles

- Tm Library Algorithms
- Optimized Load and Store routines
- Aligned Load and Store routines
- Data logging operations
- Scatter / Gather calls
- Transaction Descriptor
- Aborting a transaction, Committing a transaction
- Example Programs – Sample Codes

Source : Reference : [49], [50], [51-63],

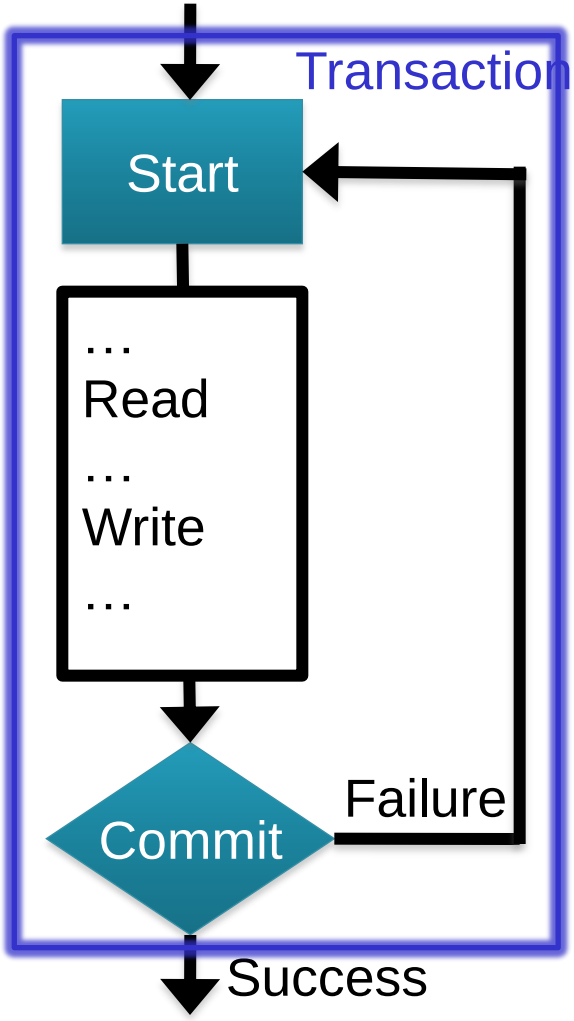
Motivation-Transactional Memory ™ Benchmarks

- Multi-core Processors -Transactional Memory (TM) : Benchmarks
- STAMP: A new benchmark suite for TM
  - 8 applications specifically for evaluating TM
  - Comprehensive *breadth* and *depth* analysis
  - *Portable* to many kinds of TMs (HW, SW, hybrid)
  - Publicly available: http://stamp.stanford.edu

Source : http://stamp.standford.edu

Multi-core /TM : Optimistic Concurrency Control

- Each core optimistically executes a transaction
- Life cycle of a transaction:
  - Start
  - Speculative execution (optimistic)
  - Build read-set and write-set
  - Commit
- Fine-grain R-W & W-W conflict detection
  - Abort & rollback



Software Transactional Memory (STM)

- Transaction – a sequence of instructions that atomically access/modify concurrent objects
- STM - A “universal” nonblocking synchronization construct
- Automatic conversion of sequential (or lock-based) code to nonblocking code
- Constant base-case overhead, but strength of nonblocking semantics
- Name coined by Shavit and Touitou (1995)

Source : Reference : [4], [6], [11],[12], [24],[25], [26], [36-63],

Transactional Memory (TM)

- Atomic and isolated execution of a group of instructions
  - All or no instructions are executed.
  - Intermediate results are not seen by other transactions.
- Programmer
  - A transaction encloses a group of instructions.
  - The transaction is executed sequentially with the other transactions and non-transactional instructions.
- TM system
  - Parallel transaction execution.
  - Register checkpoint, data versioning, conflict detection, rollback.
  - Hardware, software, or hybrid TM implementation.

Transactional Memory (TM)

- A transaction satisfies the following properties
  - Atomicity: All-or-nothing
    - On Commit: all operations become visible
    - On Abort: none of the operations are performed
  - Isolation (Serializable)
    - The transactions committed appear to have been performed in some serial order
- Additional Properties
  - Optimistic concurrency control
    - Necessary for achieving good parallel speedup
  - Non-blocking (Optional)
    - Avoid Priority Inversion
    - Avoid Convoying

Transactional Memory – Promise, Flavor , Challenges , Advantages

Software Transactional Memory

Why Software Transactional Memory?

- Unexpected delays decreases performances of locking method, besides its inherent programming difficulties.
  - Memory allocation and deallocation synchronization conflicts.
- Hardware Transactional Memory lacks the platform support, portability and delay anomalies.
- Working on a copy of the object is not good for large data structure.
- Programmable and flexible non-blocking parallel programming method is needed.

Three Challenges

- What goes in HW and what in SW?
- What are efficient algorithms and data structures for this model?
- How can we schedule transactions effectively in this model?

Challenge : TM Virtualization

- Problem
  - Limited hardware resources tuned for common cases
    - E.g. buffer size for 99% transactions
  - How do we cover uncommon cases as well?

- Cache as buffer for transactional data
  - What if cache capacity is exhausted? => space virtualization

- What if a transaction is interrupted?
  - Time virtualization

- What if transactions are nested deeply?
  - Depth virtualization

Promise of Transactional Memory

1. Easier to program
   Compose naturally
2. Easier to get parallel performance
3. No deadlocks
4. Maintain consistency in the presence of errors
5. Avoid priority inversion and convoying
6. Supports fault tolerance

```
lock(l1); lock(l2);
A = A - 10;
B = B + 10;
...
if ( error )
    recovery_code();
unlock(l1); unlock(l2);
```

```
transaction {
    A = A - 10;
    B = B + 10;
    ...
    if ( error )
        abort_transaction;
}
```

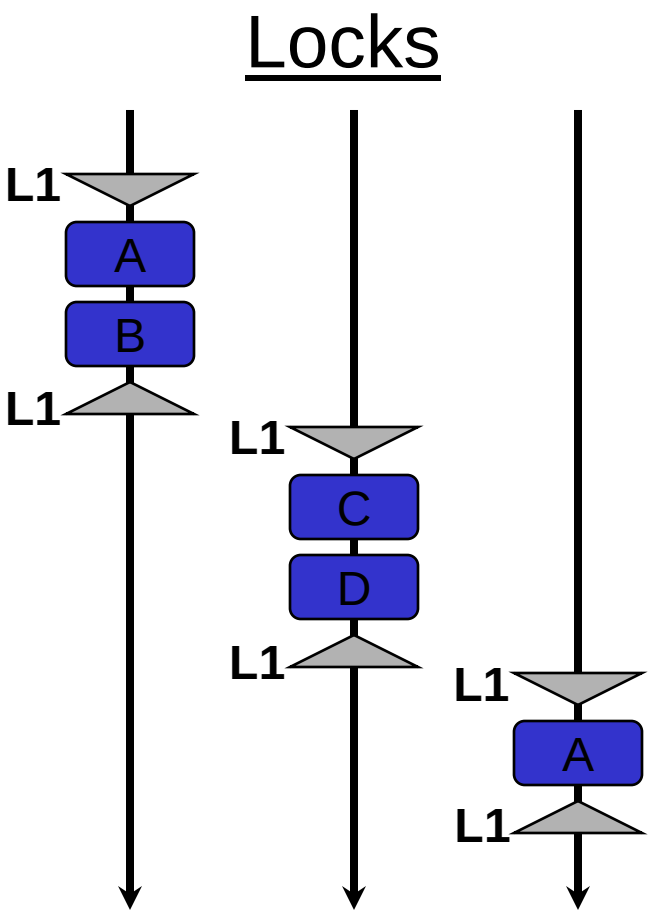
Simplify Parallel Programming

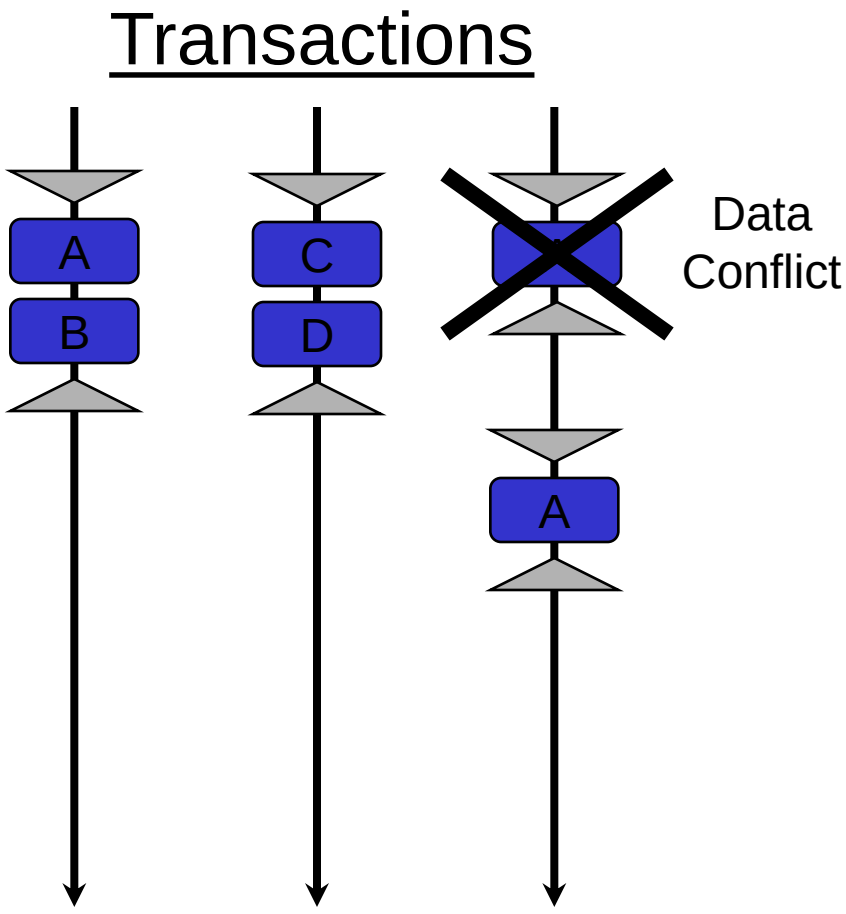
Flavors of Transactional Memory

①	Easier to program Compose naturally	Basic
②	Easier to get parallel performance	
③	No deadlocks	
④	Maintain consistency in the presence of errors	Support programmer abort
⑤	Avoid priority inversion and convoying	Support nonblocking
⑥	Supports fault tolerance	

What is work needed : Efficient support for a TM that supports all these features

Advantage 1: Performance





Serialized on Locks
Finer granularity locks helps
Burden on programmer

Optimistically execute concurrently
Abort and restart on data conflict
Automatically done by runtime

Advantage 2: Reduces Bugs

- With locks, programmers need to
  - Remember mapping between shared data and locks that guard them
    - Make sure the appropriate locks are held while accessing shared data
  - Make lock granularity as small as possible
  - Avoid deadlocks due to locks
- All of these can cause subtle bugs
- With TM, programmer does not have to deal with these problems

Other Advantages

- Allows new programming paradigms
  - Simplifies error handling
  - A new style of programming: Speculate and Verify

  Programmer can abort offending transactions
- Avoids other problems that locks suffer from
  - Priority Inversion: A low-priority thread can grab a lock and block a higher-priority thread
  - Convoying: If a thread holding a lock blocks on a high-latency event (like context-switch or I/O), it can cause other threads to wait for long periods
  - Fault Tolerant: If a process holding a lock dies, other processes will hang forever

  Runtime system can abort offending transactions

TM Benefits

- Logically sequential execution of transactions
- Optimistic concurrency control for parallel transaction execution
- No dead lock, priority inversion, and convoying
  - TM system handles pathological case
- Composability
- Error Recovery

Transactional Memory Programming Paradigm

Each thread executing a parallel region:

- Announces start of a transaction
- Executes operations on shared objects
- Attempts to commit the transaction
  - If no data race, commit succeeds, operations take effect
  - Otherwise commit fails, operations discarded, transaction restarted
- Simpler than locking!

- TM implementation on a cluster (used for Multi-Proceesors)
  - Supports both SQL and parallel scientific applications (C++)

Transactions Vs Locks

Transactional Memory Examples

Transactional Memory (TM)

- Atomic and isolated execution of instructions
  - Atomicity : All or nothing
  - Isolation : No intermediate results
- Programmer
  - A transaction encloses instructions
  - logically sequential execution of transactions

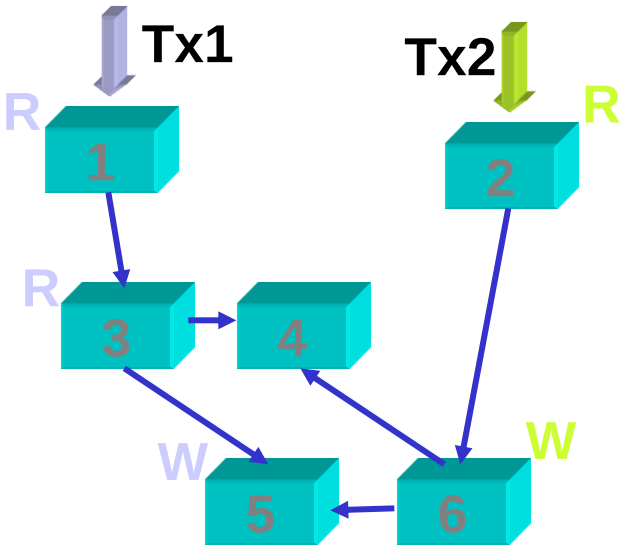
```
TX_Begin                    TX_Begin
// Instructions             // Instructions
// for Task1                // for Task2
TX_End                      TX_End
```

- TM system
  - Transactions are executed in parallel without conflict
  - If conflict, one of them is aborted and restarts

Enter Transactions

- Code segments with three features:
  - Atomicity
  - Serialization only on conflicts
  - Rollback support

TM Example



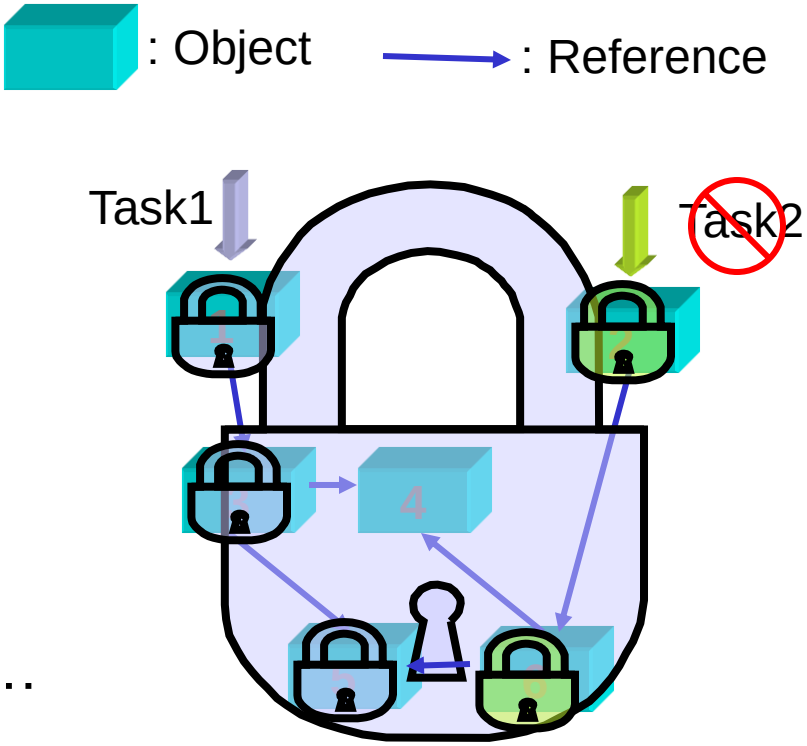
Tx 1 : $R_{[1\ 3]}$ $W_{[5]}$

Tx 2 : $R_{[2\ ]}$ $W_{[6]}$

- Data versioning
  - At TX_Begin, save register values
  - At write, save old memory values
- Conflict detection
  - Read-set and write-set per transaction
  - Conflict detection with set comparison

Locking is hard to use

- Synchronize access to shared data
- Coarse-grain locking
  - Easy to program
  - The other task is blocked
- Fine-grain locking
  - High concurrency
  - Hard to use
  - Dead lock, priority inversion, …
  - High locking overhead



Object reference graph (e.g. Java and C++)

The Challenge of Multithreaded SW

- Goal: Parallelization
- Problem: Unrestricted concurrency ➔ bugs
- Solution: Synchronization
- New problem: Synchronization
  - Tension between performance and correctness

Current Mechanism: Locks

- Locks: objects only one thread can hold at a time
  - Organization: lock for each shared structure
  - Usage: (block) ➔ acquire ➔ access ➔ release
- Correctness issues
  - Under-locking ➔ data races
  - Acquires in different orders ➔ deadlock
- Performance issues
  - Conservative serialization
  - Overhead of acquiring
  - Difficult to find right granularity
  - Blocking

Correctness, efficiency, simplicity: choose two

Transactions *versus* **Locks**

- Lock issues:
- Under-locking
- Acquires in different orders
- Blocking
- Conservative serialization

- How transactions help:
- Simpler interface
- No ordering
- Can cancel transactions
- Serialization only on conflicts

Locks ➔ simplicity/performance tension
*Transactions ➔ (potentially) simple **and** efficient*

Locks Transactions must Cooperate!

- Legacy code
- I/O
  - Nested critical section may do I/O
  - Beware low memory (page faults!)
- Critical sections may defy transactionalization
- Programmer flexibility
  - Tx performs well when actual contention is rare
  - Locks perform better when contention is high.

Challenges to Building TM Systems

Software Transactional Memory

- First try to catch the whole data it needs.
- If succeeded – compute transaction and release the data.
- If failed – release all and retry.

Transactions and Scheduling

- Transaction Restarts can waste a lot of work
- Contention Management and OS scheduler can work at cross purposes
  - HW policies avoid livelock
  - But HW policies ignore OS goals
  - e.g. timestamp
- OS requires better contention management

Software Transactional Memory

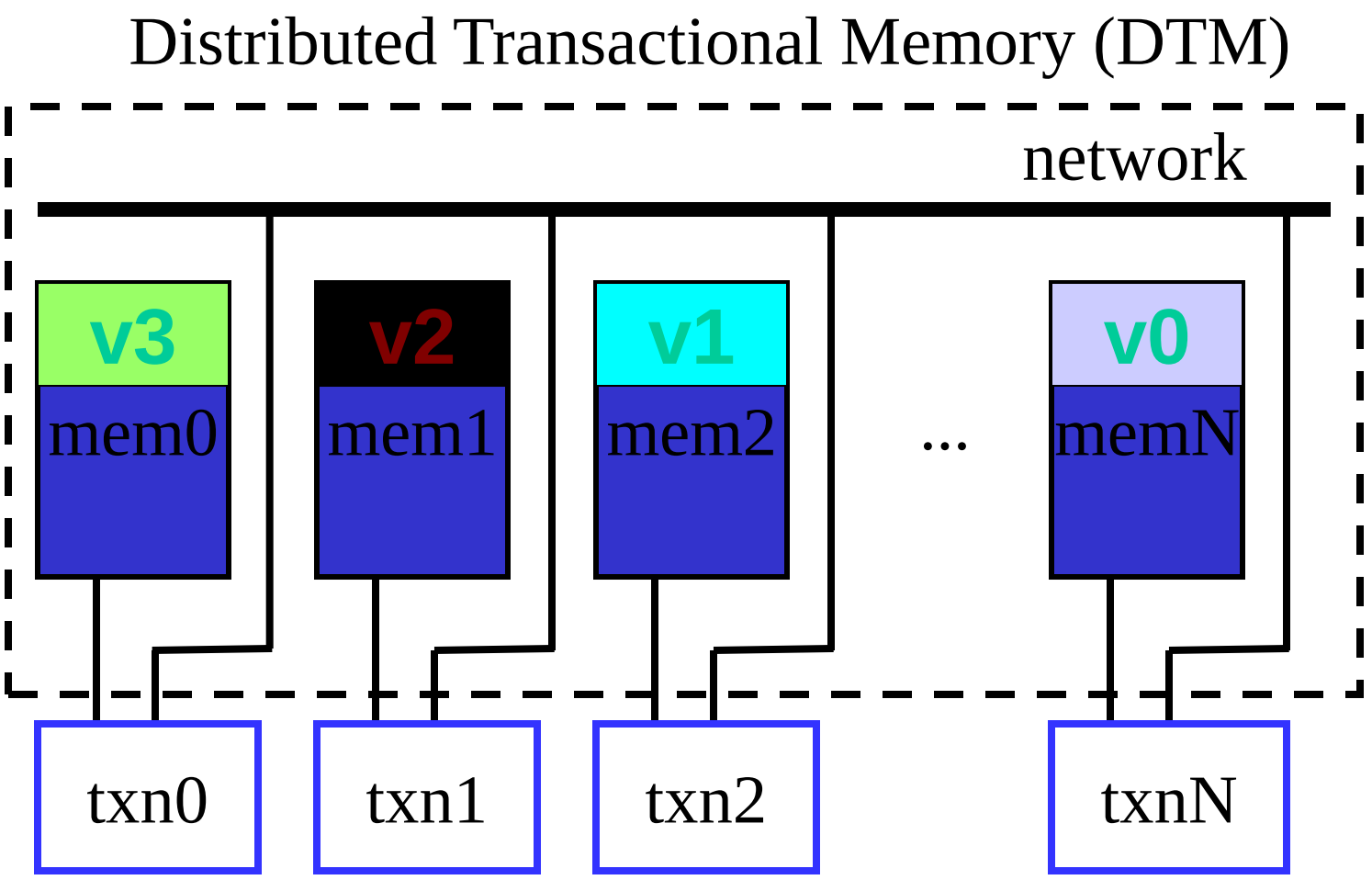
- Data set pre-acquiring
- Unintuitive programming.
- Reduces parallelism.
  - Common data structures should be acquired totally.
- Dynamic data structures are impossible.

TM System Design

- Progress in hardware and software - Growing
  - Hardware acceleration for TM is crucial for performance
    - HTM is 2 ~3 times faster than STM
  - Correctness : strong isolation

- Hardware TM
  - In the beginning
    - register checkpoint
  - At memory access
    - Set read/write bits per cache line
    - Buffer new values in cache or log old values
  - Conflict detection
    - With cache coherence protocol
    - With transaction validation protocol

Exploring Distributed Page Versions



Transactional Memory Benchmarks

Multiprocessor Benchmarks

- Benchmarks for multiprocessors
  - SPLASH-2 (1995), SPEComp (2001), PARSEC (2008)
  - Not well-suited for evaluating TM
    - Regular algorithms without synchronization problems
    - No annotations for TM
- Benchmarks for TM systems
  - Microbenchmarks from RSTMv3 (2006)
  - STMBench7 (2007)
  - Haskell applications by Perfumo et. al (2007)

STAMP Applications

Application	Domain	Description
bayes	Machine learning	Learns structure of a Bayesian network
genome	Bioinformatics	Performs gene sequencing
intruder	Security	Detects network intrusions
kmeans	Data mining	Implements K-means clustering
labyrinth	Engineering	Routes paths in maze
ssca2	Scientific	Creates efficient graph representation
vacation	Online transaction processing	Emulates travel reservation system
yada	Scientific	Refines a Delaunay mesh

TM Benchmark Suite Requirements

- Breadth: variety of algorithms & app domains
- Depth: wide range of transactional behaviors
- Portability: runs on many classes of TM systems

Benchmark	Breadth	Depth	Portability	Comments
RSTMv3	no	yes	yes	Microbenchmarks
STMbench7	no	yes	yes	Single program
Perfumo et al.	no	yes	no	Microbenchmarks; Written in Haskell

TM : STAMP Requirements

Per transaction

- Application
  - Bayes
  - Genome
  - Intruder
  - Kmeans
  - Lbayrinth
  - Ssca2
  - Vacation
  - yada

→

- No. of Instructions
- No. of Reads
- No. of Writes
- No. of Retries
- Time in Transactions (X %)

- Light-weight transactional primitives
- Transaction over context-switching
- L1 cache is sufficient
- Need for buffer space virtualization
- Multiple physical copies of data
- High memory overhead
- Prog. Lang : Java, Pthreads

TM :STAMP Requirements

- Breadth
  - 8 applications covering different domains & algorithms
  - TM simplified development of each
    - Most not trivially parallelizable
    - Many benefit from optimistic concurrency
- Depth
  - Wide range of important transactional behaviors
    - Transaction length, read & write set size, contention amount
    - Facilitated by multiple input data sets & configurations per app
  - Most spend significant execution time in transactions
- Portability
  - Written in C with macro-based transaction annotations
  - Works with Hardware TM (HTM), Software TM (STM), and hybrid TM

Using STAMP to Compare TMs

- Some other lessons we learned:
  - Importance of handling very large read & write sets (labyrinth)
  - Optimistic conflict detection helps forward progress (intruder)
- Diversity in STAMP allows thorough TM analysis
  - Helps identify (sometimes unexpected) TM design shortcomings
  - Motivates directions for further improvements
- STAMP can be a valuable tool for future TM research

55

Summary

- Challenges to building TM systems
  - Common case behavior of parallel programs
    - Extract architectural parameters for efficient TM system design
  - TM virtualization
    - Overcome the limitation of TM hardware
- Opportunity for system beyond parallel programming
  - Multithreading for dynamic binary translation
    - Fix correctness issue for DBT
  - Support for reliability, security, and fast memory snapshot
    - Improve important system metrics other than performance
- STAMP is a comprehensive benchmark suite for TM

Summary

- Transactions: Promising approach to synchronization
  - Simple interface + efficient implementation
  - Uses: optimistic lock removal, lock-free data structures, general-purpose synchronization, parallelization, ??
- Challenges
  - Implementation
  - Interface
  - OS involvement
  - I/O + rollback

Summary

- STAMP is a comprehensive benchmark suite for TM
  - Meets breadth, depth, and portability requirements
  - Useful tool for analyzing TM systems
- Public release: http://stamp.stanford.edu
  - Early adopters:
    - Industry: Microsoft, Intel, Sun, & more
    - Academia: U. Wisconsin, U. Illinois, & more
  - TL2-x86 STM

References

1. Ananian+. Unbounded Transactional Memory. (HPCA 2005)
2. Hammond+. Transactional Coherence & Consistency. (ISCA 2004)
3. Hammond+. Programming with Transactional Coherence & Consistency (TCC). (ASPLOS 2004)
4. Herlihy & Moss. Transactional Memory: Architectural Support for Lock-Free Data Structures. (ISCA 1993)
5. Moore+. Thread-Level Transactional Memory. (UW TR 2005)
6. Rajwar & Goodman. Transactional Lock-Free Execution of Lock-Base Programs. (ASPLOS 2004)
7. Rajwar+. Virtualizing Transactional Memory. (ISCA 2005)
8. http://www.cs.wisc.edu/trans-memory
9. Robert Ennals (Jan 2006). Software Transactional Memory Should Not Be Obstruction-Free. Technical Report Nr. IRC-TR-06-052. Intel Research Cambridge Tech Report.
10. K. Fraser. Practical Lock-Freedom. Technical Report UCAM-CL-TR-579, Cambridge University Computer Laboratory, February 2004.
11. Tim Harris , Keir Fraser. Language support for lightweight transactions. Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications, October 26-30, 2003, Anaheim, California, USA.
12. Maurice Herlihy, Victor Luchangco, Mark Moir, and William N. Scherer III. Software Transactional Memory for Dynamic-Sized Data Structures. ACM Symposium on Principles of Distributed Computing (PODC): 92-101, 2003.
13. Maurice Herlihy , Victor Luchangco. Distributed computing and the multicore revolution. ACM SIGACT News, v.39 n.1, March 2008.
14. Virendra J. Marathe and William N. Scherer III and Michael L. Scott (Oct 2004). Design Tradeoffs in Modern Software Transactional Memory Systems. In: Proceedings of the 7th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers. Houston, TX.
15. N. Shavit and D. Touitou. Software transactional memory. Distributed Computing, Special Issue(10): 99-116, 1997.
16. William N. Scherer III and Michael L. Scott (Jul 2004). Contention Management in Dynamic Software Transactional Memory. In: Proceedings of the ACM PODC Workshop on Concurrency and Synchronization in Java Programs. St. John's, NL, Canada. In conjunction with PODC'04.

References

17. Ennals' blocking STM: Robert Ennals. Efficient Software Transactional Memory. Intel Research Cambridge Technical Report: IRC-TR-05-051, 2005. PRAM:
18. S. Fortune and J. Wyllie. Parallelism in Random Access Machines. In Proceedings of the 10th Annual Symposium on Theory of Computing, pages 114-118, 1978.
19. Phillip B. Gibbons , Yossi Matias , Vijaya Ramachandran. Can shared-memory model serve as a bridging model for parallel computation?. Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures, p.72-83, June 23-25, 1997, Newport, Rhode Island, United States. 1997
20. P. B. Gibbons. A more practical PRAM model. Proceedings of the first annual ACM symposium on Parallel algorithms and architectures, p.158-168, June 18-21, 1989, Santa Fe, New Mexico, United States. 1989
21. Popular message-passing old models:
22. David Culler , Richard Karp , David Patterson , Abhijit Sahay , Klaus Erik Schauser , Eunice Santos, Ramesh Subramonian , Thorsten von Eicken. LogP: towards a realistic model of parallel computation. ACM SIGPLAN Notices, v.28 n.7, p.1-12, July 1993.
23. Leslie G. Valiant. A bridging model for parallel computation. Communications of the ACM, v.33 n.8, p.103-111, Aug. 1990.
24. Memory allocation in multi-core:
25. Andrei Gorine, Konstantin Knizhnik. Tackling memory allocation in multicore and multithreaded applications. MCObject LLC, May 2006. Available on the internet from http://www.embedded.com/columns/showArticle.jhtml?articleID=188101359 , May 2006
26. Voon-Yee Vee , Wen-Jing Hsu. A Scalable and Efficient Storage Allocator on Shared Memory Multiprocessors. Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '99), p.230, June 23-25, 1999.
27. P.R. Wilson, M.S. Johnstone, M. Neely, and D. Boles. Dynamic storage allocation: A survey and critical review. In H.G. Baker, editor, Proceedings of International Workshop on Memory Management (IWMM'95), volume 986 of Lecture Notes in Computer Science, pages 1-116, Kirnoss, Scotland, Sept. 1995.

References

28. Mark Moir, Sun Micro System Laboratories, USA *Technical Perspective – Transactional Memory in the Operating System,* Communications of the ACM September 2008, Volume 51, No. 9, page 82
29. Christopher J, Rossbach, Hany E Ramadan, Owen S Hofmann, Donlad E Porter, Aditya Fhandari, and Emmett Witchel - *TxLinux and MetaTM : Transactional Memory and the Operating System* – Communications of the ACM September 2008, Volume 51, No. 9, page 83-91
30. Adi-Tabatabai, A. R. Lewis, B.T. Menon, V. Murphy B. R, Saha B and Shpeisman T, *Compiler and runtime Support for efficient software transaction memory* IN PLDI, June 2006
31. Carstrom B. McDonald, A . Chafi, H. Chung J., Cao Minh C. Kozyrakis, C and Olukotun K, *The Atmos Transactional programming Language,* in PDLI, June 208
32. Engler D and Ashocraft K Racer-X, *Effective static detection of race conditions and deadlocks* in SOSP 2003
33. Herlihy M and Moss J. E , *Transactional memory Architectural Support for lock-free data structures* in ISCA, May 1993
34. Larus J.TR and Rajwar R *Transactional Memory*, Morgan & Claypool 2008
35. Rajwar R and Goodman J *Transactional lock-free execution of lock-based programs* In ASPLOS 2002
36. Ramadan H, Rossbach C and Wiltchel E, *The Linux Kernel A challenging workload for Memory* In workshop on Transactional Memory Workloads June 2006
37. Herb Sutter, *Use Lock Hierarchies to Avoid Deadlock* Dr. Dobb's Journal, January 2008, page 67-69 http://www.ddj.com
38. Andrews, Grogory R. *Foundations of Multithreaded, Parallel, and Distributed Programming,* Boston, MA : Addison-Wesley, 2000
39. Butenhof, David R *Programming with POSIX Threads ,* Boston, MA : Addison Wesley Professional, 1997
40. Culler, David E., Jaswinder Pal Singh *Parallel Computer Architecture - A Hardware/Software Approach* , San Francsico, CA : Morgan Kaufmann, 1999
41. Grama Ananth, Anshul Gupts, George Karypis and Vipin Kumar Introduction to Parallel computing, Boston, MA : Addison-Wesley, 2003
42. Intel Corporation, *Intel Hyper-Threading Technology, Technical User's Guide*, Santa Clara CA : Intel Corporation Available at : http://www.intel.com 2003

References

43. Pacheco S. Peter, Parallel Programming with MPI, , University of Sanfrancisco, Morgan Kaufman Publishers, Inc., Sanfrancisco, California, 1992
44. Kai Hwang, Zhiwei Xu, Scalable Parallel Computing (Technology Architecture Programming), McGraw Hill New York., 1998
45. Michael J. Quinn, Parallel Programming in C with MPI and OpenMP McGraw-Hill International Editions, Computer Science Series, McGraw-Hill, Inc. Newyork, 2004
46. Marc Snir, Steve Otto, Steyen Huss-Lederman, David Walker and Jack Dongarra, *MPI-The Complete Reference: Volume 1, The MPI Core, second edition* [MCMPI-07]., 1998
47. William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir *MPI-The Complete Reference: Volume 2, The MPI-2 Extensions, 1998*
48. *Intel Software Tool Kit* http://www.intel.com
49. *Intel Transactional Memory compiler and Runtime Application -binary Interface, Nov 2008,* ***http://www.intel.com***
50. Lawrence Crowl (Google), Yossi Lev (Brown University), Victor Lunchangco (Sun Microsystems), Mark Moir (Sun Microsystems), Dan Nussbaum (Sun Microsystems) *Integrating Transactional Memory into C++,* TRANSACT 2007
51. Idan Igra, William N Scherer III, Mark Noir, Victor Lnchangco, Mauricd Herlihy,*Topics in Reliable distributed computing*, Techion Nov 2008
52. Maurice Herlihy *Transactional Memory* - Brown University
53. Sanjeev Kumar, Miachael Chu, Chrisopher Hughes, Partha Kundu (Intel Labs) *Hybrid Transactional Memory* http://www.intel.com
54. Virendra J. Marathe, William N. Scherer III, and Michael L. Scott, Department of Computer Science, University of Rochester, *Design Tradeoffs in Modern Software Transactional Memory Systems*
55. *Kaloian Manassiev, Madalin Mihailescu and Cristiana Amza, University of Toronto, Canada, Exploiting Distributed Version Concurrency in a Transactional Memory Cluster*
56. *Michael L. Scott, Sandhya Dwarkadas, William N. Scherer III,Virendra J. Marathe, Michael F. Spear, Arrvindh Shriraman,Vinod Sivasankaran, Hemayet Hossain, Luke Dalessandro,Athul Acharya, Chris Heriot, David Eisenstat, and Aaron Rolett, University of Rochester,Exploiting Distributed Version Concurrency in a Transactional Memory Cluster,Contributions summary, TRAMP Workshop, February 2007*

References

57. http://www.cs.rochester.edu/research/synchronization/rstm/ *Rochester Software Transactional Memory*
58. Chris Rossbach, Owen Hofmann, Don Porter, Hany Ramadan, Aditya Bhandari, Emmett Witchel, University of Texas at Austin, TxLinux : *Managing Transactional Memory in an Operating System*
59. http://stamp.stanford.edu/ (*Stanford Transactional Applications for Multi Processing*)
60. JaeWoong Chung, Michael Dalton, Hari Kannan,Christos Kozyrakis, Computer Systems Laboratory, Stanford University,*Thread-Safe Dynamic Binary Translation using Transactional Memory*
61. Chí Cao Minh, JaeWoong Chung,Christos Kozyrakis, Kunle Olukotun, Standford Transactional Applications for Multi Processing, http://stamp.stanford.edu 15, September 2008
62. Brian D. Carlstrom, Computer Systems Laboratory, Stanford University, http://tcc.stanford.edu *Programming with Transactional Memory*
63. Colin Blundell, E Christopher Lewis, Milo *Martin, University of Pennsylvania,* Transactional Memory Overview

Thank You

Any questions ?