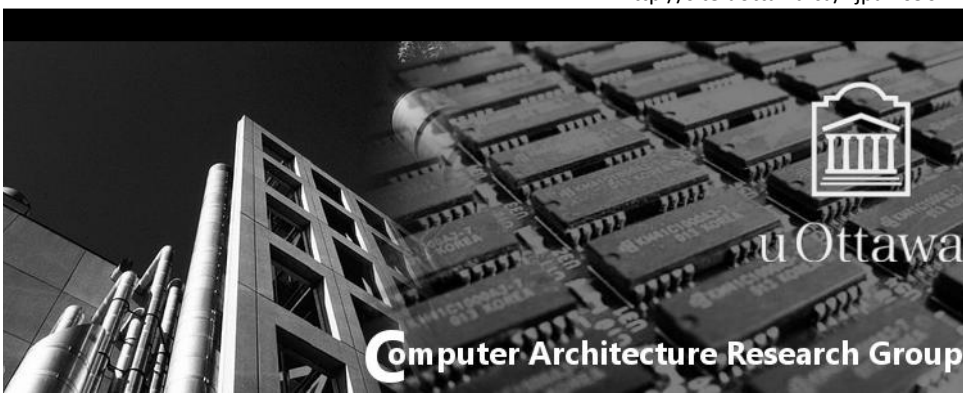
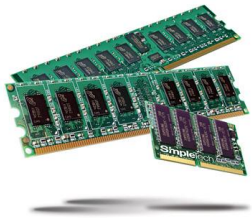
An Introduction to Transactional Memory

Jonathan Parri
jparri@uottawa.ca
http://site.uottawa.ca/~jparr090



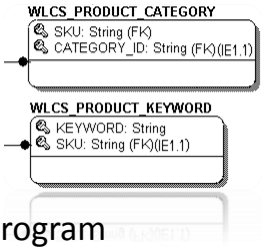
Prelude

How can we simplify parallel programming by allowing groups of load and store operations to execute in an atomic fashion?



Transactions

- Transactions first used as database unit abstractions.
- Transactions appear indivisible and instantaneous to an observer.
  - Follow ACID Properties[Micro98]
    - Atomicity
    - Consistency
    - Isolation
    - Durability



- The differences between database and program models has brought about the transactional memory term for programs. [Lomet77] (Old Idea ~1977!)

Transactional Memory

- The idea of transactional memory lay dormant until 1993.
  - Hardware Supported Transactional Memory [Herlihy93]
    - Shows efficient lock-free synchronization.
    - Extension to any multiprocessor cache-coherence protocol.
  - Oklahoma Update [Stone93]
    - Providing multiple reservations to remove critical sections from concurrent programming.
    - An update atomically changes all shared variables or none of them.
- Transactional Memory provides an abstraction for coordinating concurrent read and writes. Currently, coordination is the responsibility of the programmer with rudimentary constructs (Java, C/C++).

Software Transactional Memory

- Software is more flexible than hardware
- Software is easier to change
- Software Transactional Memory models have fewer limitations (address width etc…)

Software Transactional Memory

- Software Transactional Memories require a mechanism for associating concurrency-control metadata with the locations that a program is accessing.[Harris2010]
  - **Object-based system**: Metadata is held with each object the program allocates.
  - **Word-based system**: Metadata is associated to each word of storage.

Software Transactional Memory

- As transactions can be rolled-back a system of **undo/redo logs** are required.
  - Lazy version management
    - Redo log of values that will be written to memory if the transaction commits.
  - Eager version management
    - Logging granularity
    - Offsets versus Addresses
    - Type of value

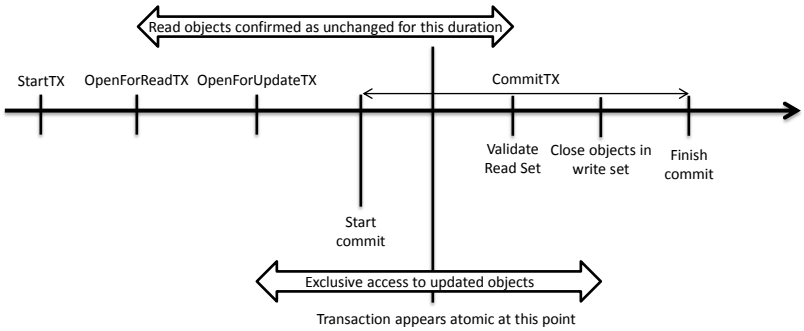
Software Transactional Memory

- Lazy version management that writes to n locations can take up to O(n) operations for each read.
- We can improve this with:
  - Auxiliary Look-up tables [Harris2006]
  - Bloom filter to maintain summary of write-set [Burton70]
  - Metadata can provide links to red- log entries for transactions that are currently writing to locations described by metadata.

Software Transactional Memory

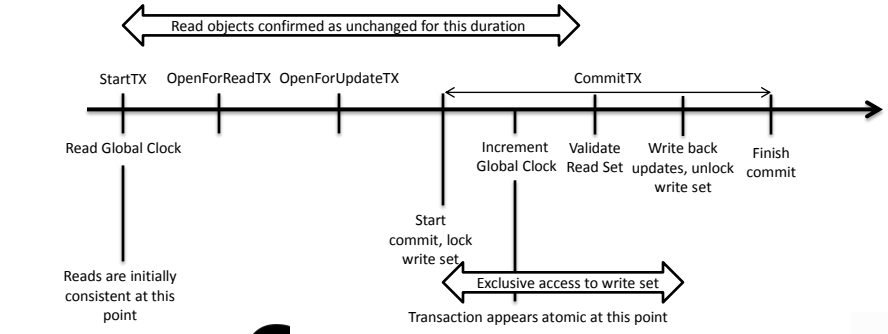
- Lock-Based Systems with Local Version Numbers [Dice2006]
  - Dynamically acquires locks for writes. Reads implemented by checking per-object version numbers during a validation stage.
  - Version numbers incremented locally and independently whenever an update is committed.
  - Does not provide opacity during transactions.
  - Does not provide conflict detection between transactional and non-transactional accesses to memory locations.

Software Transactional Memory



Software Transactional Memory

- Lock-Based Systems with a Global Clock [Dice2007]
  - Clock incremented on a process-wide basis.
  - Record Read Version and Write Version



Software Transactional Memory

- Lock-Based Systems with Global Metadata [Olszewski2007]
  - Only shared metadata are global constructs with no individual locks or versions.
  - No individual locks or version numbers
  - Avoid space overhead and cache pressure
  - Reduce number of atomic operation involved in running and committing a transaction
  - Requirements:
    - Sparse access to global meta data to reduce memory contention
    - Must detect conflicts in terms or actual locations that a transaction accesses (instead of executing transactions sequentially)
    - Solution= Bloom Filter or value based validation

Software Transactional Memory

- Non-Blocking Software Transactional Memory
  - All transactions appear to take place atomically.
  - Threads cannot prevent another thread from accessing a memory location used by the primary thread.
  - Primary thread must ensure:
    - It will see all other threads updates or none of them
    - If primary thread attempts conflicting memory access it will be detected and resolved automatically

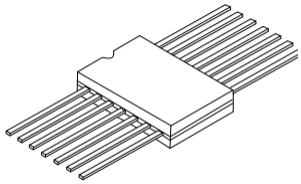
Software Transactional Memory

- Non-Blocking Software Transactional Memory with Dynamic Software Transactional Memory
  - [Herlihy2003] Programmer need not specify memory locations that require access for transaction in advance.
  - Explicit contention management module
  - Work led to many later projects with the basis of:
    1. Dynamically associating objects with a transaction descriptor
    2. Defining logical contents of an object as a set of metadata and snapshots
    3. Using immutable shared data to allow threads to determine an object's logical value (avoids locking)

Software Transactional Memory

- Taxonomy of DSTM design choices [Spear2006]:
  1. When are objects acquired for writing?
  2. Are readers visible to writers?
  3. How are objects acquired by transactions?
  4. How do objects appear when they are inactive?
  5. Which non-blocking progress property is provided?

Hardware Transactional Memory



- Lower overhead
- Better power usage (debatable)
- Less invasive
- Provide strong encapsulation without the needing to change the way non-transactional memory accesses occur
- Effective for languages without dynamic compilation and garbage collection

Hardware Transactional Memory

- Similar to their software counterparts hardware transactional memories must:
  - Identify locations for transactional accesses
  - Manage read and write-sets of transactions
  - Detect and resolve data conflicts
  - Manage architectural register state
  - Commit or abort transactions

Hardware Transactional Memory

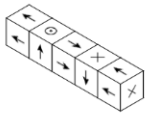
- Transactional memory access must be identified using available instruction set extensions:
  - Explicitly Transactional → Provide software with new memory instructions to show which memory accesses are to be made transactionally.

    Examples:
    - Herlihy and Moss HTM [Herlihy93]
    - Oklahoma Update [Stone93]
    - Advanced Synchronization Facility from AMD [Christie2010]
  - Implicitly Transactional → Software need only allow specifications of transaction boundary. All memory accesses within the boundary are transactional.

    Examples:
    - Speculative Lock Elision[Rajwar2001]
    - Rock Processor from SUN [Sun2009]
    - Azul [Azul2008] (seems as though they have moved away from HTM with claims of <10% performance gains, now focusing on Java acceleration!!!)

Hardware Transactional Memory

- Must track transaction's read-set and buffer transaction's write-set
- Advantageous over software since hardware has caches and buffers already in place for tracking memory accesses
- These mechanisms are extended so their exists no overhead



Hardware Transactional Memory

- Extending the existing data cache is a popular choice to track read-sets as done in:
  - Speculative Lock Elision[17]
  - Rock Processor from SUN [18]
- Typically an extra bit to each cache line entitled the "read bit"
  - Transactional read operations set the bit showing that the cache line is part of the transaction's read-set
  - Can clear all read bit flags in one go

Hardware Transactional Memory

Granularity of conflict detection is constrained to cache lines!

Hardware Transactional Memory

- Keeping track of write-sets is more difficult
- Data cache can again be extended by adding a speculatively written state for involved addresses [Blundell2007].
- Due to base processor architectures, latest transactional updates are automatically forwarded to subsequent read operations within the transaction
- If the cache is used to track the write-set then we must ensure that a unique copy of a line is not lost

Hardware Transactional Memory

- Some solutions do not modify the cache but instead add dedicated buffers to track read and write-sets like AMD's Advanced Synchronization Facility
- Others use a hybrid approach
  - Data cache used to track read-set
  - Store buffer used to track write-set

Hardware Transactional Memory

- [Zilles2007] Results of evaluating effectiveness or data cache tracking for both read and write-sets:
  - 32KB 4-way set associative data cache
  - 64 byte cache lines
  - Showed that transaction could usually support up to 30000 instructions that operate on hundreds of cache lines

Hardware Transactional Memory

- Some new methods are moving away from cache line granularity and offer hardware support for the software object construct [Khan2008]
- Transaction management is integrated with object translation buffer in object-based execution models (seems more theoretical with object-aware memory architectures)

Hardware Transactional Memory

- We must detect data conflicts
  - Checking whether read and write-sets for concurrently executing transactions overlap and conflict
  - Common cache coherence mechanisms are used on local buffers to track read and write-sets to show conflicts **M(O)ESI**

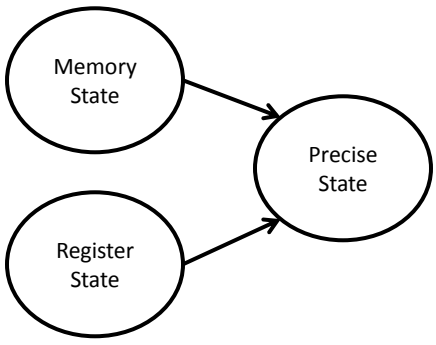
Hardware Transactional Memory

- After determining that a conflict has occurred a resolution must take place, conflict resolution
  - Transaction receives conflicting request is aborted
  - Control is transferred to software
  - Software handler executes and either re-executes transaction or performs contention management

New and important research field!

Hardware Transactional Memory

- So far most work has looked at the memory state but omitted the state of the register file
- If we abort a transaction we must return the register bank back to a previous state
  1. Rely on software for recovery
  2. Use shadow registers to restore state
  3. Exploit architectures with speculative execution
  4. Hybrid

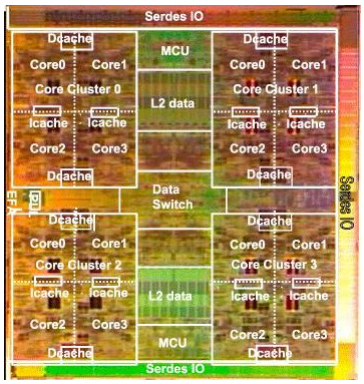


Hardware Transactional Memory

- When committing a transaction, the update must be visible to all other processors
  - If a buffer is used:
    - Write permissions for all transactional addresses must be obtained
    - Blocking of requests from other processors
    - Draining of the store buffer into the data cache
  - If the data cache is used:
    - Requires a new cache state array design
    - Read and write-sets should must be operated on instantaneously

Sun's Rock Architecture [Chaudhry2009]

- Implicitly Transactional
  - Instruction set extensions
    1. `chkpt<fail-address>` instruction used to specify an address to go to following an abort
    2. All operations between a `chkpt` and `commit` instruction are transactional
- Data cache used to track read-set
- Store buffer used to track write-set
- L2 cache required to store all addresses inside a transaction
  - When `commit` is executed, the L2 cache locks all cache lines corresponding to the tracked store addresses
  - The stores drain the store buffer, update the cache, then release the lock preventing conflicting access during a commit sequence



Sun's Rock Architecture

- A flash-copy mechanism is used to save the architectural register state
  - Register file is divided into two segments, static memory cells (SRAM) and active registers
  - Checkpoints are stored in static portion
  - Active portion maintains register file used for execution
  - It takes 1 cycle to restore register state from static to active portion
- A special `cps` register exists which identifies why a transaction was aborted

AMD's Advanced Synchronization Facility [AMD2009]

- 2009 proposal by AMD to be added to the x86 instruction set
- Explicitly Transactional
  - Instruction set extensions
    1. `SPECULATE` instruction used to begin a transaction
    2. `COMMIT` instruction is used to mark the end
    3. Control returns to is the speculative region aborts and sets appropriate processor flags
    4. Keeps speculating until commit occurs
    5. `LOCK` added to memory accesses that need to be performed transactionally
    6. Dedicated registers used to perform multi-word compare-and-swap like instructions
    7. Able to recover stack pointer to relies on software for the rest of the registers

AMD's Advanced Synchronization Facility

Example of compare and swap (DCAS) on two independent memory locations using ASF from AMD:

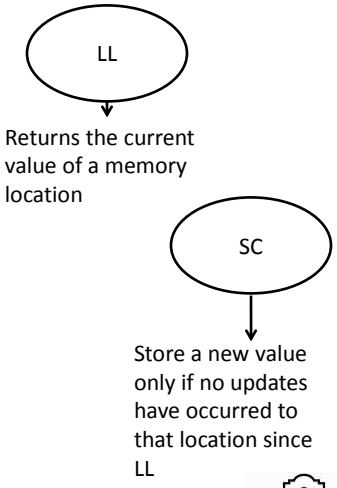
```
DCAS:
          MOV R8, RAX
          MOV R9, RBX
retry:
          SPECULATE ; Speculative region begins
          JNZ retry ; Page fault, interrupt, or contention
          MOV RCX, 1 ; Default result, overwritten on success
          LOCK MOV RAX, [mem1] ; Specification begins
          LOCK MOV RBX, [mem2]
          CMP R8, RAX ; DCAS semantics
          JNZ out
          CMP R9, RBX
          JNZ out
          LOCK MOV [mem1], RDI ; Update protected memory
          LOCK MOV [mem2], RSI
          XOR RCX, RCX ; Success indication
out:
          COMMIT ; End of speculative region
```


Simulators

- (2010) PTLSim augmentation with AMD64 architecture extension for transactions. (cycle accurate!)
  - http://www.amd64.org/research/multi-and-manycore-systems.html
- (2009) Deuce STM provides support for transactions within the JVM.
  - http://www.deucestm.org/
- (2010) Dresden TM Compiler supporting transactions in C/C++.
  - http://tm.inf.tu-dresden.de
- (2008) IBM XL C/C++ for Transactional Memory Compiler introduces pragma based atomic specification in C/C++.
  - http://amino-cbbs.sourceforge.net/
- (2009) Intel C++ STM Compiler with C++ extensions.
  - http://software.intel.com/en-us/articles/intel-c-stm-compiler-prototype-edition/
- (2008) Java library implementing STM with partial re-execution of failed transactions.
  - http://wen.ist.utl.pt/~joao.cachopo/jvstm
- (2007) MetaTM is a hardware transactional memory simulator for the Virtutech Simics platform.
  - http://www.metatm.net

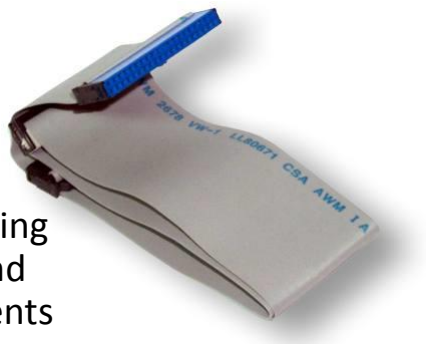
LL/SC and Transactional Memory

- Load-link/store-conditional (LL/SC)
  - Implement a lock-free atomic read-modify-write operation
  - Typically considered the most basic transactional memory support
  - Operates on data that is the size of a native machine word (bound)



Conclusion

- Transactions are a new programming abstraction
- Software and hardware transactions complement each other more than they compete
- Transactions are believed to be the future programming mechanism of multicore and multiprocessing environments



References

[Micro98] "Gray to be Honored With A. M. Turing Award This Spring". Microsoft PressPass. 1998-11-23. http://www.microsoft.com/presspass/features/1998/11-23gray.mspx. Retrieved 2010-11-10.

[Lomet77] David B. Lomet. Process structuring, synchronization, and recovery using atomic actions. In *ACM Conference on Language Design for Reliable Software*, pages 128–137, March 1977.

[Herlihy93] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: architectural support for lockfree data structures. In *ISCA '93: Proc. 20th Annual International Symposium on Computer Architecture*, pages 289–300, May 1993.

[Stone93] Janice M. Stone, Harold S. Stone, Phil Heidelberger, and John Turek. Multiple reservations and the Oklahoma update. *IEEE Parallel & Distributed Technology*, 1(4):58–71, November 1993.

[Harris2010] T. Harris, J. Larus. R. Rajwar. Transactional Memory, Morgan & Claypool, 2010, pp 103

[Harris2006] Tim Harris, Mark Plesko, Avraham Shinnar, and David Tarditi. Optimizing memory transactions. In *PLDI '06: Proc. 2006 ACMSIGPLANConference on Programming Language Design and Implementation*, pages 14–25, June 2006.

[Burton70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[Dice2006] David Dice and Nir Shavit. What really makes transactions faster? In *TRANSACT '06: 1st Workshop on Languages, Compilers, and Hardware Support for Transactional Computing*, June 2006. 47, 108, 116

[Dice2007]Dave Dice and Nir Shavit. Understanding tradeoffs in software transactional memory. In *CGO '07: Proc. International Symposium on Code Generation and Optimization*, pages 21–33, March 2007.

[Olszewski2007]Marek Olszewski, Jeremy Cutler, and J. Gregory Steffan. JudoSTM: a dynamic binary rewriting approach to software transactional memory. In *PACT '07: Proc. 16th International Conference on Parallel Architecture and Compilation Techniques*, pages 365–375, September 2007.

[Herlihy2003]Maurice Herlihy, Victor Luchangco, Mark Moir, and William N. Scherer III. Software transactional memory for dynamic-sized data structures. In *PODC '03: Proc. 22nd ACM Symposium on Principles of Distributed Computing*, pages 92–101, July 2003.

[Spear2006] Michael F. Spear, Virendra J. Marathe,William N. Scherer III, and Michael L. Scott. Conflict detection and validation strategies for software transactional memory. In *DISC '06: Proc. 20th International Symposium on Distributed Computing*, September 2006.

References

[Stone93]Janice M. Stone, Harold S. Stone, Phil Heidelberger, and John Turek. Multiple reservations and the Oklahoma update. *IEEE Parallel & Distributed Technology*, 1(4):58–71, November 1993.

[Christie93]Dave Christie, Jae-Woong Chung,Stephan Diestelhorst,MichaelHohmuth,MartinPohlack,

Christof Fetzer, Martin Nowack,Torvald Riegel, Pascal Felber, Patrick Marlier, and Etienne Riviere. Evaluation of AMD's advanced synchronization facility within a complete transactional memory stack. In *EuroSys '10: Proc. 5th ACMEuropean Conference on Computer Systems*, April 2010.

[Rajwar2001] Ravi Rajwar and James R. Goodman. Speculative lock elision: enabling highly concurrent multithreaded execution. In *MICRO '01: Proc. 34th International Symposium on Microarchitecture*, pages 294–305, December 2001.

[Sun2009] http://www.opensparc.net/pubs/preszo/08/RockHotChips.pdf

[Azul2008] http://www.azulsystems.com/blog/cliff-click/2009-02-25-and-now-some-hardware-transactional-memory-comments

[Blundell2007]Colin Blundell, Joe Devietti, E. Christopher Lewis, and Milo M. K. Martin. Making the fast case common and the uncommon case simple in unbounded transactional memory. *SIGARCH Computer Architecture News*, 35(2):24–34, 2007.

[Zilles2007]Craig Zilles and Ravi Rajwar. Implications of false conflict rate trends for robust software transactional memory. In *IISWC '07: Proc. 2007 IEEE INTL Symposium on Workload Characterization*, September 2007.

[Khan2008]Behram Khan, Matthew Horsnell, Ian Rogers, Mikel Luján, Andrew Dinn, and IanWatson. An object-aware hardware transactional memory. In *HPCC '08: Proc. 10th International Conference on High Performance Computing and Communications*, pages 93–102, September 2008.

[Chaudhry2009] Shailender Chaudhry, Robert Cypher, Magnus Ekman, Martin Karlsson, Anders Landin, Sherman Yip, Håkan Zeffer, and Marc Tremblay. Rock: A high-performance Sparc CMT processor. *IEEE Micro*, 29(2):6–16, 2009.

[AMD2009]http://developer.amd.com/assets/45432-ASF_Spec_2.1.pdf