Transactional Memory: The Surprising Complexity of a Simple Idea

Michael L. Scott

at Carnegie Mellon University
6 November 2008

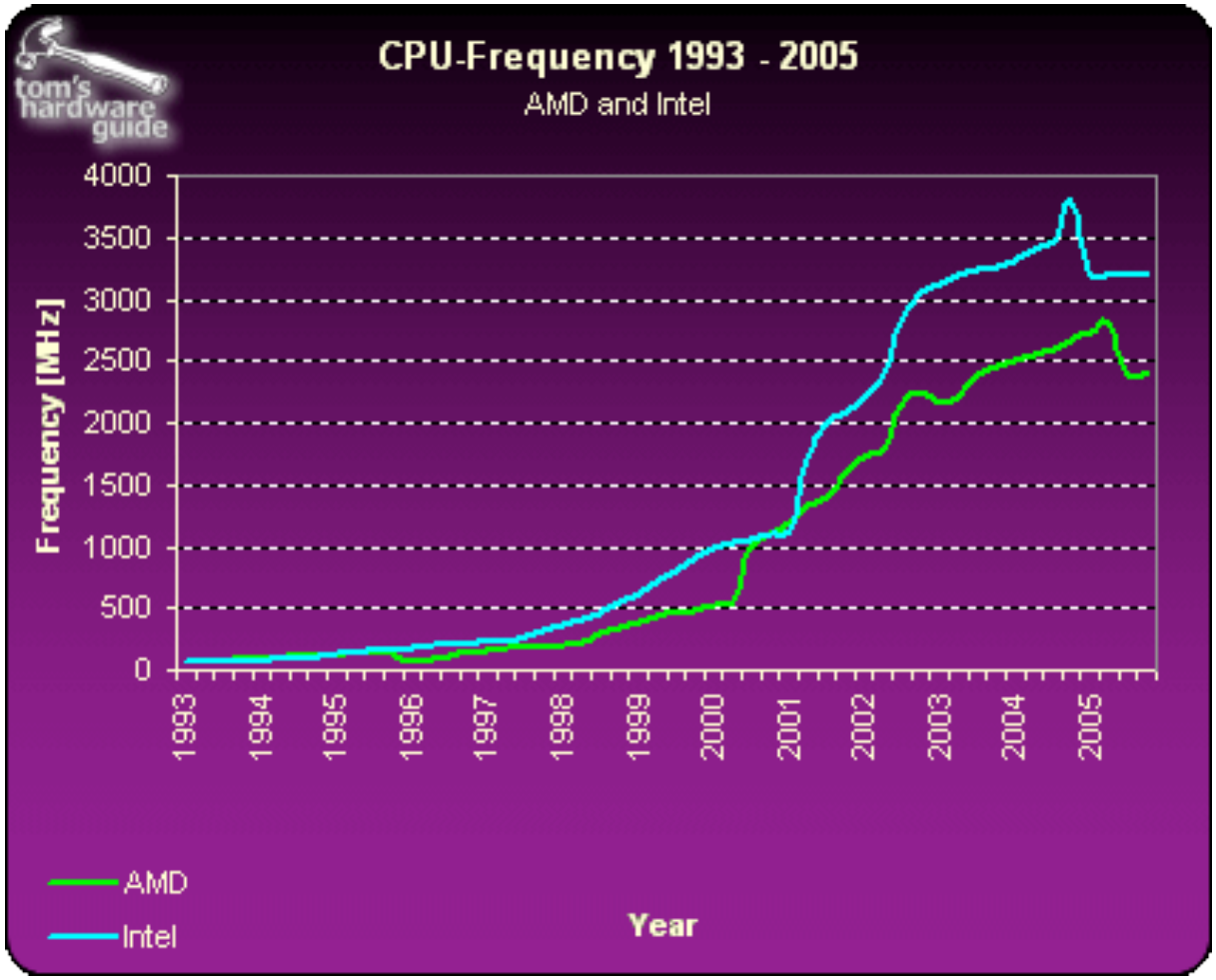
Moore's Law: The Free Ride Is Over

- Smaller feature size allowed higher clock rates
  - » better performance BUT ALSO more energy
    - Wattage $\propto$ mm^2 × clock rate
  - » and we ran out of cooling
- Smaller feature size also allowed more tricks on the die at a given clock rate
  - » superpipelining, superscalar and OOO issue, speculation
  - » better performance without more energy
  - » but we ran out of tricks

1995 v. 2005

http://www.tomshardware.com/2005/11/21/the_mother_of_all_cpu_charts_2005/page2.html

Density ∝ performance only if year ≤ 2004



http://www.tomshardware.com/2005/11/21/the_mother_of_all_cpu_charts_2005/

Enter Multicore

- Multiple processors (*cores*) on each chip
  - » maybe ratchet back the clock and (esp.) the tricks (forgo diminishing returns)
- But this is no longer invisible to the user of traditional programming languages
  - » programs have to be *multithreaded*

The Coming Crisis

- Parallelism common in high-end scientific computing
  - » done by experts, at great expense
- Also common in Internet servers
  - » "embarrassingly parallel"
- Has to migrate into the mainstream
  - » programmers not up to the task

slurmed.com

The Traditional Model

- Explicit threads, with *locks* for mutual exclusion
- In use since the mid 1960s
- Well understood, but hard to use correctly
  - » acquire wrong lock; forget to release
  - » deadlock due to ordering
  - » priority inversion
  - » inopportune preemption
  - » convoying
  - » lack of *composability*
- Performance/complexity tradeoff

The "Transactional Religion"

- Butler Lampson: every good idea in operating systems came from the database community

⇨ Lightweight transactions (atomic, consistent, isolated)
Herlihy & Moss [1993], Shavit & Touitou [1995], ...,

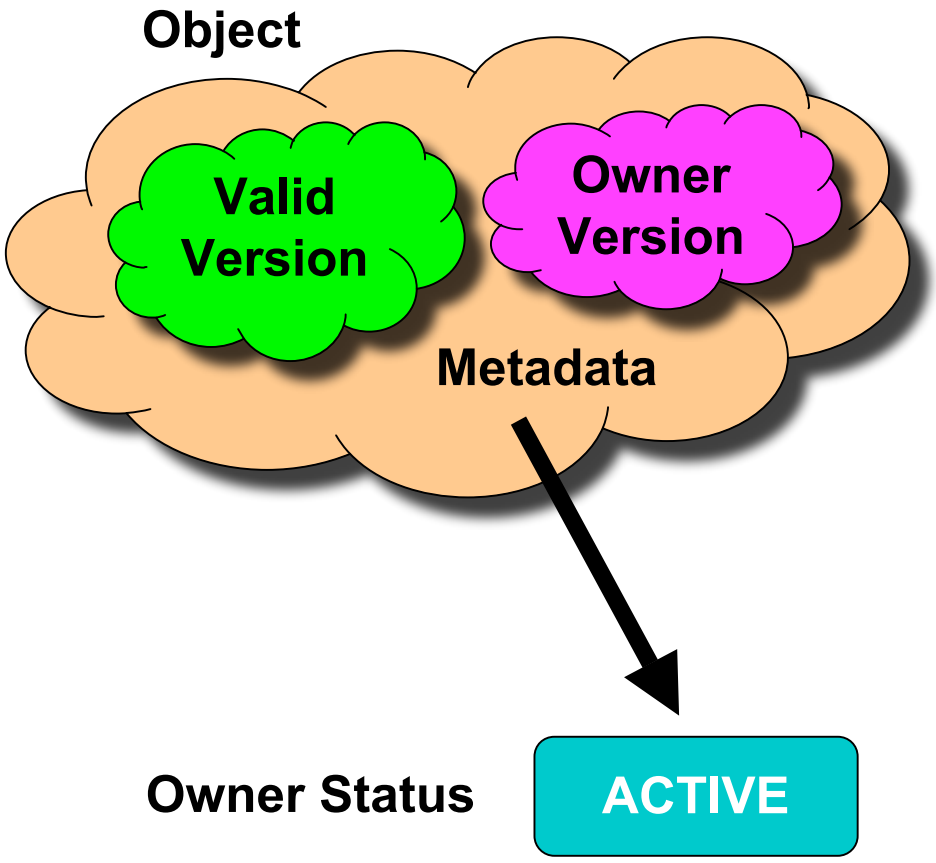
A Simple Idea

- User labels atomic sections

```
atomic {
        ...
}
```

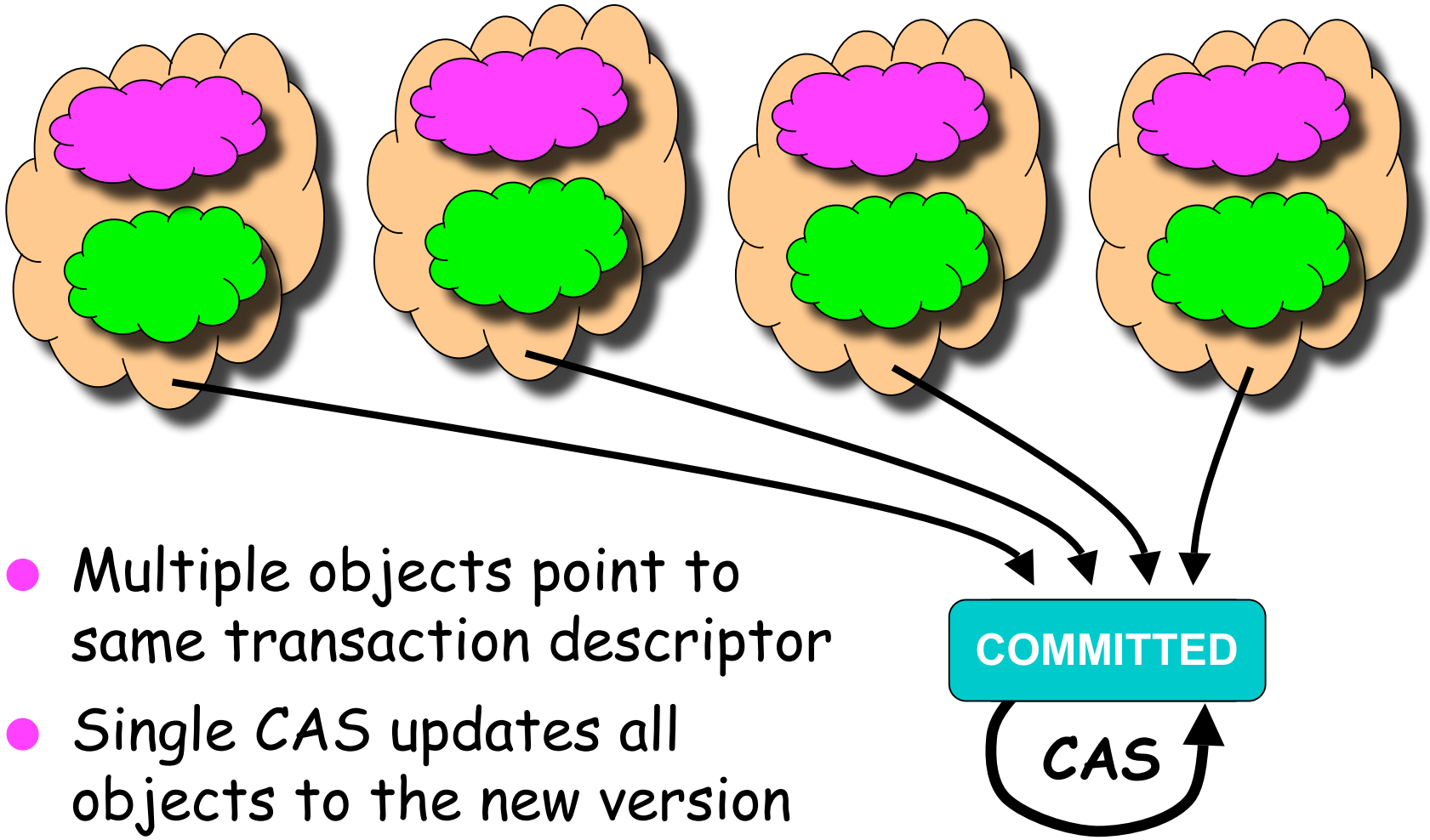
- Underlying system ensures atomicity, isolation, and consistency; executes in parallel when possible
- Implementation expected to be speculative; back out and re-try on conflict
  - » requires HW or SW checkpointing / logging
  - » doing both at Rochester; focus here on SW

STM, abstractly



- Only owner can change the object
- Only one transaction can own the object at a time
- Until the owner commits, everyone sees Valid Version

Atomic Commit



- Multiple objects point to same transaction descriptor
- Single CAS updates all objects to the new version

Big Conceptual Benefits

- Avoid deadlock, priority inversion — *composability*
- Tolerate thread failures (w/ nonblocking implementation)
- ✶ Eliminate the tradeoff between concurrency and clarity:
  - » system's job, not the programmer's, to figure out what can run in parallel
  - ⇨ the complexity of coarse-grain locks with (most of) the performance of fine-grain locks

Major Implementation Issues

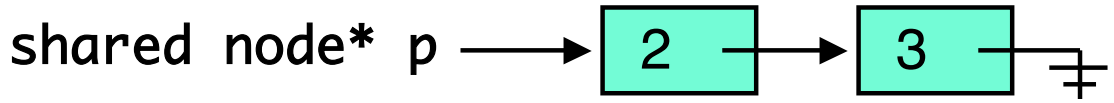
- Conflict detection / consistency preservation
  - » eager v. lazy
  - » visible readers v. incremental validation v. timestamps v. Bloom filters
- Buffering: cloning v. redo v. undo
- Lock-based v. nonblocking (OF? LF?)
- Conflict resolution
  - » Who wins? Who loses?
  - » What progress guarantees can we make (if any)?

- Explosion of papers over the past 5 years

Semantic Complications

- Nesting
- Condition synch. (retry)
- Exceptions (need language support!)
- Irreversible ops / inevitable txns
- Interaction w/ locks, NB data structures
- Ability to "leak" info from aborted txns
- Privatization and publication

A Privatization Puzzle




```
shared int n = 0;

A: atomic {                          B: atomic {
     my_node = p->next                    if (p->next)
     p->next = nil                          p->next->val = 4
     i = n                                n = 1
   }                                    }
   print i, my_node->val
   delete my_node
```

- What might this code print?
  - » 0 3 (A first)
  - » 1 3 ??
  - » bus error ??
  - » 1 4 (B first)
  - » 0 4 ??

The Publication/Privatization Problem

- SW txns serialize by reading & writing metadata
- Want to avoid that OH when poss. —> private use
  - » Delaunay mesh creation app: 95+% private
- But Bad Things can happen at the public/private boundaries
  - » delayed cleanup @ privatization
  - » doomed txns @ privatization
  - » early reads @ publication

What Semantics Do We Want?

- Memory models suggests: appearance of sequential consistency for properly synchronized programs
- But what is "properly synchronized" for TM?
  - static data partition
  - global phase consensus
  - privatizing / publishing transactions (explicit?)
  - private / transactional races ("strong isolation")?
- Is the language implementation required to catch bad programs? Statically?
- If not, are there constraints on what bad programs can do?
  - Cf Java and C++ MMs

My Personal Take

- Static partition is too restrictive
- Transactional / nontransactional races are bugs
  - » Cf DRF
  - » if *r* and *T* conflict, a transaction in *r*'s thread must intervene
- As in Java, consequences of bugs are limited — program can't "catch fire"
  - » in particular, no out-of-thin-air reads

Database Semantics

- Serializability (S)
  - » Observed history must be equivalent to (same ops, same results) some serial history (no overlapping txns) with the same thread subhistories
- Strict Serializability (SS)
  - » Additionally, if 2 txns (of different threads) do not overlap in the observed history, they must appear in the same order in the serial history
  - » Motivation: prevent threads from using outside events to observe txns in the "wrong" order — plane ticket example

Single Lock Atomicity

- (SLA) Transactions behave "as if" they acquired a single global lock
  - » Equivalent to SS:
    - serial txn order ≡ lock acquisition order
    - locks force order wrt nontxnal accesses w/in threads
  - » Widely considered too expensive to implement
    - At begin_txn, must ensure no peer has prefetched published data
    - At end_txn, must ensure all previous txns have cleaned up, and all doomed txns aborted

Relaxing Order

- Multi-lock semantics [Menon et al.'07]
  - » separate reader-writer lock for every datum
  - » several alternative locking protocols; relax requirement for serializability
- But
  - » Explains behavior in terms of (multiple) locks — which txns were supposed to replace!
  - » Abandons serial order for txns — arguably the key to success in the DB world
- Alternative proposal [OPODIS '07]
  - » Define semantics in terms of ordering (Cf: Java, C++)
  - » Keep transactions serial; make txnal-nontxnal ordering optional

The Bigger Picture: Keep the Simple Case Simple

- Partition shared and private data
- Atomic is simply atomic; data is just data ("no asterisks")
- Compiler has to figure out a lot
  - » Inevitability for irreversible operations
  - » Static inference of always-private data
  - » Automatic cloning for transactional and private contexts
- If you need more, turn the page
  - » condition sync
  - » privatization
  - » leaking
  - » interoperation w/locks
- But if you don't, don't

TM Work at Rochester

- RSTM suite of TM implementations
  - » all major options from the literature
  - » dozens of back-end variants
  - » uniform API based on C++ smart pointers and templates
    - good for experimentation; not for naive users
- Exploration of
  - » implementation basics: conflict detection and resolution, buffering [CSJP'04, LCR'04, PODC'05 (2), DISC'06, SPAA'08]
  - » inevitability and retry mechanisms [TRANSACT'08, ICPP'08, PODC'08]
  - » privatization [PODC'07, ICPP'08]
  - » hardware acceleration [TRANSACT'06, PPoPP'07, ISCA'07, SPAA'07, ASPLOS'08, ISCA'08]
  - » nonblocking implementations [PODC'05, DISC'05, TRANSACT'06, PPoPP'08]
  - » application studies [NGS'07, PODC'07, IISWC'07, TRANSACT'07]
  - » semantics [SCOOL'05, TRANSACT'06, DISC'07, OPODIS'08]

Status of the Field

- HW support in Azul and Sun processors
- SW projects underway at Intel, Sun, Microsoft, and IBM (at least)
- SW performance results are mixed — a win in some cases, a loss in others — real benefits are in ease of use
- Will (in my opinion) succeed at simplifying the creation of parallel data structure libraries
- Not yet clear how much more will succeed
- Is not a panacea!

Ongoing Work

- Runtime implementation issues: private use, irreversibility, conflict detection and contention management [Mike Spear]
- Formal semantics, with privatization
- Language integration
- Compiler implementation [Luke Dalessandro]
- Hardware acceleration [Arrvindh Shriraman]
- Application development

- Longer term 

Where Will all the Threads Come From

- Programming idioms / design patterns
  - » e.g., futures, p-o iterators, dataflow, ...
- Higher-level abstractions
  - » map/reduce/scan, ...
- Speculative parallelization
  - » manual or automatic
  - » transactions for automatic detection and recovery from uncommon data races
- (Your silver bullet here)

www.cs.rochester.edu/research/synchronization/