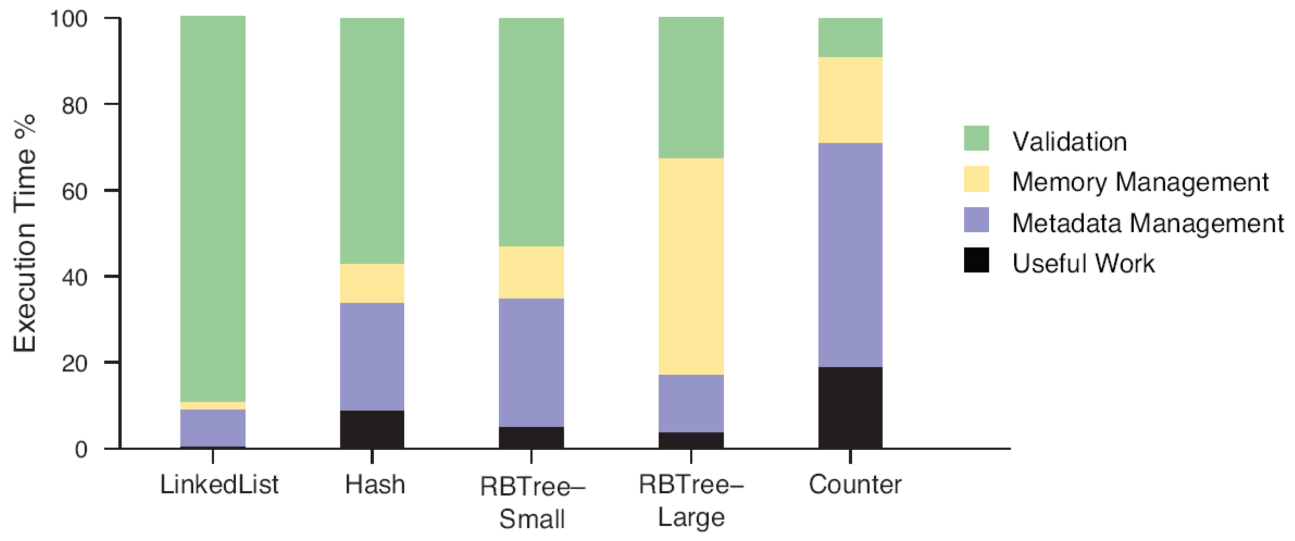
Lecture Notes (Tuesday 29 April, 2008)

With all this talk about software transactional memory, how well does it perform?

NZSTM for example, is about 5 times slower than acquiring a coares-grained lock on a single thread!

What does it spend all its time on? RSTM, for example:-



Cost breakdown for RSTM on a single processor, for five different microbenchmarks (Shiram et al, 05)

Open question: have we reached a ceiling, an inherent upper limit in how well STM can perform in software?

Any similarities other technology in history? Garbage collection for example? (Recommended reading: Grossman, http://www.cs.washington.edu/homes/djg/papers/analogy_oopsla07.pdf)

Similarities:
- Both claim to make programming easier
- Both were (are) kind of controversial
- Both add complexity to the environment to support it, and potentially incur some overhead

Unsubstantiated conjectures::
- GC didn't need hardware support to succeed
- GC took a while to become mainstream
- GC performs as well, sometimes even better, than good manual memory management

What's being done now? A good repository of TM related work, old and new:-
http://www.cs.wisc.edu/trans-memory/biblio/index.html

Correctness:

A system that promises reliability is expected to be correct. How do we know that it is?

Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.   - Dijstra

Either prove it mathematically,, or model it and prove that the model is correct. The first approach might be difficult while the second one might be incomplete...

An example of modeling:
http://www.cag.lcs.mit.edu/~rinard/paper/scool05.pdf

Benchmarking

How can we test transactional memory, and know that it's actually useful? Benchmark it...

The problem with TM benchmarks; they're either contrived and don't represent real work applications. The ones that are based on real applications convert critical sections to transactions; however, critical sections were designed without transactional memory in mind. Therefore they're probably small and carefully tweaked for being used with locks...

Programmers won't use TM until they see representative benchmarks, and representative benchmarks are difficult to get without having programmers write programs with TM. (chicken & egg problem)

Current projects involving benchmarks:
Standford's STAMP: http://stamp.stanford.edu/
STMBench7: http://lpd.epfl.ch/site/research/tmeval
Linux Kernel: http://www.cs.utexas.edu/~rossbach/pubs/wtw06ramadan.pdf

Support for Transactional Memory

SNZI: Scalable NonZero Indicators

Why use a counter when all you need is a boolean value?
http://research.sun.com/scalable/pubs/PODC2007-SNZI.pdf

PhTM: Phased Transactional Memory

Change what mode the transactional memory runs in depending on the current environment.
http://research.sun.com/projects/scalable/pubs/TRANSACT2007-PhTM.pdf

Other Issues

Transaction Nesting
Checkpointing
Language Support
Privatization

Older Issues

Blocking vs Nonblocking
Visible vs Invisible Reads
Strong vs Weak Atomicity (Isolation)
Contention Management