

What Will It Take for STM to be Useful?

Michael L. Scott, Sandhya Dwarkadas,
Virendra J. Marathe, Michael F. Spear,
Arrvindh Shriraman, Vinod Sivasankaran,
Hemayet Hossain, Luke Dalessandro, Aaron Rolett

University of Rochester
www.cs.rochester.edu/research/synchronization/

March 2007

Starting Assumptions

- Simplicity the overriding goal
(if it's messy, why not just use locks?)
- Performance important, but secondary
 - » *modest* steady slowdown is acceptable
 - » pathological slowdown is not, even if rare
- Transition has to be gradual
 - » must interoperate with existing code, inc. locks and ad hoc nonblocking data structures
 - » must run on existing HW (maybe faster on new HW)
- Nonblocking semantics desirable but not essential

TM Contributions

- STM systems (mostly nonblocking) in Java & C++
- Pluggable contention management
- Metadata organization
- Conflict detection and validation
- Formal semantics
- Synchronization across transactions
- Bag-of-tasks transaction scheduling
- Hardware acceleration (RTM, RTM-Lite)
- Please see [summary/pub list on TRAMP website](#)

Qualitative Conclusions to Date

- Simplicity matters in practice (Delaunay app.)
 - » library is not enough — need language support
 - » any messiness should be optional
- Adaptation is essential
 - » adjust policy to match workload characteristics
 - » have separate fast code path
- Privatization is essential
 - » private code must suffer negligible overhead
- Consistency is harder than it looks
 - » sandboxing is more than type safety and signal handlers
 - » incremental validation an appealing alternative, esp. for unmanaged code, and esp. with HW support

Current Research Tracks

- Library-based (C++) STM for impl. experiments
 - » good test bed, though not for naive users
 - » metadata layout, conflict detection, contention mgmt., privatization, adaptation, ...
- Hardware acceleration [ISCA'07, SPAA'07]
 - » alert on update, programmable data isolation; other ideas in the works
 - » leave policy to software
 - » preference for general-purpose mechanisms (alert-on-update good for lots of things)
- Application development
- Language/compiler design (mostly future work)
 - » ask me later for my wish list

Privatization

- Allow data to move in/out of transactional world
 - » producer/consumer, spatial partitioning, ...
- Conceptually simple
- Two implementation problems
 - » private code fails to see committed but not-yet-cleaned-up updates
 - » doomed but not-yet-aborted transaction sees private updates and takes erroneous action
- Several technical solutions [paper in submission]
 - ➡ how to present to user?

Transactional Sharing Models

- Contract between the user & the system
 - » Cf. programmer-centric memory consistency models
 - » ideally enforced by compiler
- Transactions *appear* to be strongly isolated if programmer follows the rules
 - » static partition — **too restrictive**
 - » partition within global consensus phases
 - e.g. via barriers
 - » privatizing transactions
 - multiple possible implementations
 - » strong isolation
 - probably too expensive for software — **overkill**

TRANSACT'07

**The Second ACM SIGPLAN
Workshop on Transactional Computing**

To be held in conjunction with **PODC 2007**
Portland, Oregon, August 16, 2007

Submission deadline: April 15, 2007

www.cs.rochester.edu/meetings/TRANSACT07/