Software Transactional Memory

Panacea or Pandora's Box?

Christos Kozyrakis

Assistant Professor of EE & CS
Stanford University

http://csl.stanford.edu/~christos

My Message In a Nutshell

- Transactional Memory (TM)
  - Atomic access to shared state
  - User declares parallelism ⇒system manages
- The bad news
  - Still in relatively early R&D stages
  - TM cannot solve all problems around parallelism
- The good news
  - TM is an very powerful tool
  - Uses go beyond concurrency management
- Needed: systems ⇒users & apps ⇒feedback

Programming with TM

- Integration with HL languages & compilers
  - Java/C# (PLDI'06, PPoPP'07), OpenMP (PACT'07), …
  - Declarative approach to sharing data
- Rich features beyond atomic{}
  - Iterators, variable types, retry, abstract locks, …
  - Speculative parallelization
- Still missing: truly HL parallel model(s)
  - E.g., DBs use atomic{} but program SQL
  - Deal with finding concurrency, locality, coordination, and scalability in addition to atomicity

TM and I/O

- Can have some I/O model within TM but…
  - Special libraries, restricted APIs, …
- Better idea: system-level transactions
  - TM is one of the many system resources
  - Use transactional mechanisms with other resources
    - FS ⇒LFS, Net ⇒message queues, DBMS
  - Coordinate transactions across multiple resources
  - Easier-to-use & flexible model
- Can this ever work?
  - Look at IBM's Quicksilver project from the 80s

Beyond Concurrency

- When you build TM, you build
  - Atomicity, consistency, & isolation
  - Mechanisms to monitor memory accesses
- Great building blocks for
  - Debugging (deterministic replay, watchpoints, parallel bookmarks)
  - Profiling & tuning (contention & locality monitoring)
  - Security (isolated execution, canaries)
  - Fault-tolerance (undo on error, checkpointing)
  - …
- These can be killer apps for TM
  - Equally important for programmability

Can TM use HW Support?

- Yes
  - HW reduces TM overheads for common case
  - Don't forget that performance is ****the**** motivation for parallel programming
  - Low overhead is particularly important when TM is used extensively for non-concurrency purposes
- But
  - HW provides basic mechanisms, SW sets policies
    - E.g., SigTM design at ISCA'07
  - HW should have flexible semantics (see ISCA'06)
  - TM apps should work even without HW support