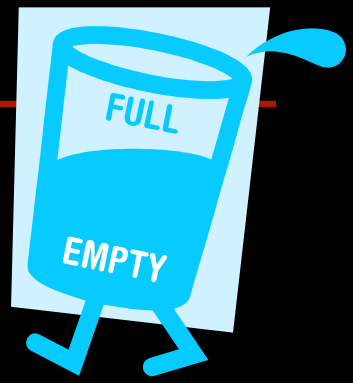
~~Potential Show-Stoppers for Transactional Synchronization~~



Christos Kozyrakis

Computer Systems Lab
Stanford University

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

Ok, the base TM ideas look good; what's next?

Christos Kozyrakis

Computer Systems Lab
Stanford University

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

1. Apps & User Studies

- Are we really simplifying parallel programming?
  - Let's write new apps or get feedback from others
- What are the common cases and pattern?
  - This is what we'll make simpler, faster, …
  - Are we sure TM is sufficient to address all of them?
- Casting lock-based apps in TM is dangerous
  - Will fine-grain, rare transactions be common?

2. atomic{} is a primitive, not a parallel programming model

- DB users program SQL, not atomic{}
- Need truly high-level programming models
  - Simple & declarative like SQL, Mapreduce, …
  - atomic{} will be critical in implementing them
  - But it will probably take more than atomic{}
    - Primitives for finding concurrency and handling locality, coordination, scheduling, balance…
- Prog. environment = language + tools + libs
  - Use TM to build better debugging/tuning tools
  - See talk in next session for the libs issue

3. Atomicity ≠ Coordination

- TM is not a hammer for every nail
- Lots of work on forcing coordination into TM
  - Open-nesting, escape actions, non-isolated transactions, dependent transactions, …
  - Use semantics get really ugly, really quickly
  - Is it worth it? What do we expose to user and how?
- Simpler idea: use TM for what it is
  - Transactions = atomicity + isolation
  - Combine with other primitives to address other problems

4. Transactional memory & I/O

- TM is not a hammer for every nail
  - We can have restricted I/O within TM but…
- Better idea: make TM work with other transaction resources in the system
  - DB, LFS, message queues, …
  - System-level manager coordinates user transaction across all resources
  - Easier-to-use, flexible model with some restrictions
- Can this ever work?
  - Look at IBM's Quicksilver project

5. Beyond concurrency control

- Atomicity & isolation are generally useful
  - For debugging, profiling, checkpointing, exception handling, garbage collection, security, speculation …
- These may be TM's initial "killer apps"
- But they also change the requirements
  - Cheap transactions for pervasive use
  - "All transactions, all the time"

Miscellaneous TM Issues

- Language support: YES
- Compiler support: YES
- HW support: YES
- Strong atomicity: YES
- Contention management: YES
- Compensating actions: YES
- High-level concurrency control: YES
- …

- ***9am*** **panels: NO**