Semantics for Transactional Languages

Michael L. Scott

ASPLOS PC “mini symposium”
21 October 2011

thoughts inspired by a series of wonderful students
A Simple Idea [Herlihy & Moss ’93]

- User labels atomic sections
  ```
  atomic {
    ...
  }
  ```

- Underlying system ensures atomicity; executes in parallel when possible (speculation)

- Motivations
  - performance (via lock elision)
  - simplicity: as easy as coarse-grain locks
Since Then

- Surge of interest ~2003, with the multicore revolution and with breakthroughs in HW (UWisc, UIUC) and SW (Sun, Cambridge)
- Scores of papers & systems; at least a dozen active groups; TRANSACT 7 coming up
- Through the trough of disillusionment?
A Fleeting Opportunity

• HTM is coming
  » Azul, Sun Rock, AMD ASF
  » IBM has announced for Blue Gene/Q
  » ... ?

• STM for backward compatibility, fallback on HW overflow

• Language support essential

• Narrow window in time to “get the semantics right”
Outline

• Assertions
  » atomicity is central
  » speculation is an implementation issue (only)
  » small transactions are what matter
  » privatization is essential
    – necessary for correctness
    – solves the problem of legacy synchronization

• Open Questions
  » non-transactional reads and writes
  » big transactions, integration with system transactions
  » relationship to “deterministic parallel programming”
Memory Model

• Transactional sequential consistency (TSC)
  » ideal but expensive: global total order on accesses
    – consistent w/ program order \( <_p \)
    – w/ each transaction contiguous — atomic

• Strict serializability (SS)
  » txns globally totally ordered wrt one another
  » also ordered wrt preceding & following accesses of same thread (though those accesses aren’t necessarily globally ordered wrt one another)

• Transactional Data-Race Freedom (TDRF)
  » if all conflicting accesses are ordered by SS
  » then the program appears to be TSC
Strong Isolation Is a Non-Issue

• Hard to explain to the programmer
  » what is a memory access?

• Heavy performance penalty in STM

• Only matters in racy programs
  » constrains the behavior of buggy code
  » less than you want (TSC); more than you need to build what you want (TSC given TDRF)
  » may be useful for debugging, but a good race detector is better
Opacity Is a Semantic Non-Issue

- Aborted transactions do not appear in (language-level) histories
- Opacity is simply one end of the implementation spectrum: validate at every read
- Sandboxing is the other end: validate before every “dangerous” operation (and periodically)
- Some very promising implementations in the middle: delayed/out-of-band validation
  » ask me later!
Privatization Is Essential

- Definition: transaction $T$ with history prefix $P$ privatizes datum $D$ if
  - $\exists$ extensions of $P$ in which a first access to $D$ after $P$ occurs in different threads
  - $\forall$ extensions of $P+T$, the first access to $D$ after $P+T$ occurs in the same one thread

- Crucial for performance with STM

- Solves the problem of legacy synchronization
  - locking is privatization — acquire and release are small atomic blocks
Transactions $\neq$ Critical Sections

L.acquire()               atomic {
...\equiv\ldots
L.release()               }

L.acquire()            atomic { ... }
...\equiv\ldots
L.release()            atomic { ... }
Open Questions
Non-transactional Accesses

• Want reads for, e.g., ordered speculation, high-performance hybrid TM
  » clearly important at the HTM ISA level
  » not clear whether needed/wanted at language API level

• Want writes out of aborted txns for debugging
  » again, clearly important at the HTM ISA level
    – and probably more useful if immediate
  » probably important at the language level too
    – not as clear that these need to be immediate

• Immediate writes, and writes in aborted txns, a challenge for the memory model
Atomicity and Determinism

- See our paper at DISC’11
  - languages/idioms that guarantee all abstract executions will be “equivalent” in some well-defined sense
- Independent split-merge an obvious foundation for language-level determinism
- Atomic commutative [+associative] ops an obvious extension
- Is there anything else?
  - atomic event handlers, perhaps?
The Bottom Line: Keep It Simple!

- Atomicity is central
- Speculation is an implementation issue (only)
- Small transactions are what matter
- Privatization is essential (and solves the problem of legacy synchronization)
www.cs.rochester.edu/research/synchronization/

Thanks to Bill Scherer, Virendra Marathe, Mike Spear, Luke Dalessandro, Li Lu, ...