On Reconciling Hardware Atomicity, Memory Models, and __tm_waiver

Sean White and Michael Spear
Lehigh University

WTTM 2010
On the Meaning of Non-Transactional Stores

Sean White and Michael Spear
Lehigh University

WTTM 2010
Best Effort HTM is Tricky

• Lots of caveats
  – **Capacity** limitations
  – No context switching implies duration limitations
  – Need software fallback

• Automatic conflict resolution
  – Invisible, **immediate** aborts
  – Even thread-safe libraries may not be safe

• Cannot execute “poison” operations / **syscalls**
  – Very bad things happen
Possible Solution: “Helper” Threads

• Always-available non-transactional context

• Alleviates many BEHTM limitations
  – Syscalls
  – Logging/pure functions
  – Asynchronous library calls

• May require inter-thread communication
  – Synchronous library calls (e.g., malloc)

• May simplify BEHTM design
Communicating Within Transactions

atomic {
    y = 42;
    // “x = malloc(64)”
    send(malloc, 64)
    x = receive()
    ...
}

Transactional Store

Send Request (Implies a non-transactional store)

Blocks Until Response

Must happen before the store to y!
Memory Model

• This ordering breaks strong memory models!

• What should a non-transactional store mean?

• Rock: forbid this ordering
  – Good for TLS, bad for TM… preserves TSO

• ASF: allow this ordering
  – Bad for TLS, good for TM… breaks x86 memory model
Our Proposal

• Have 3 types of stores inside of transactions
  – Transactional
  – Memory model-preserving non-transactional
  – Immediate non-transactional

• Pros: can do TLS and let TM use helper threads

• Cons: complexity
  – How to implement in HW?
  – How to implement in SW?
    • Not really __tm_waiver...
Conclusions

• Meaning of non-transactional stores is underspecified

• Meaning of __tm_waiver is underspecified

• They are probably incomparable

• Lots of power in relaxing store ordering within transactions
  – But **without** relaxing it all the time