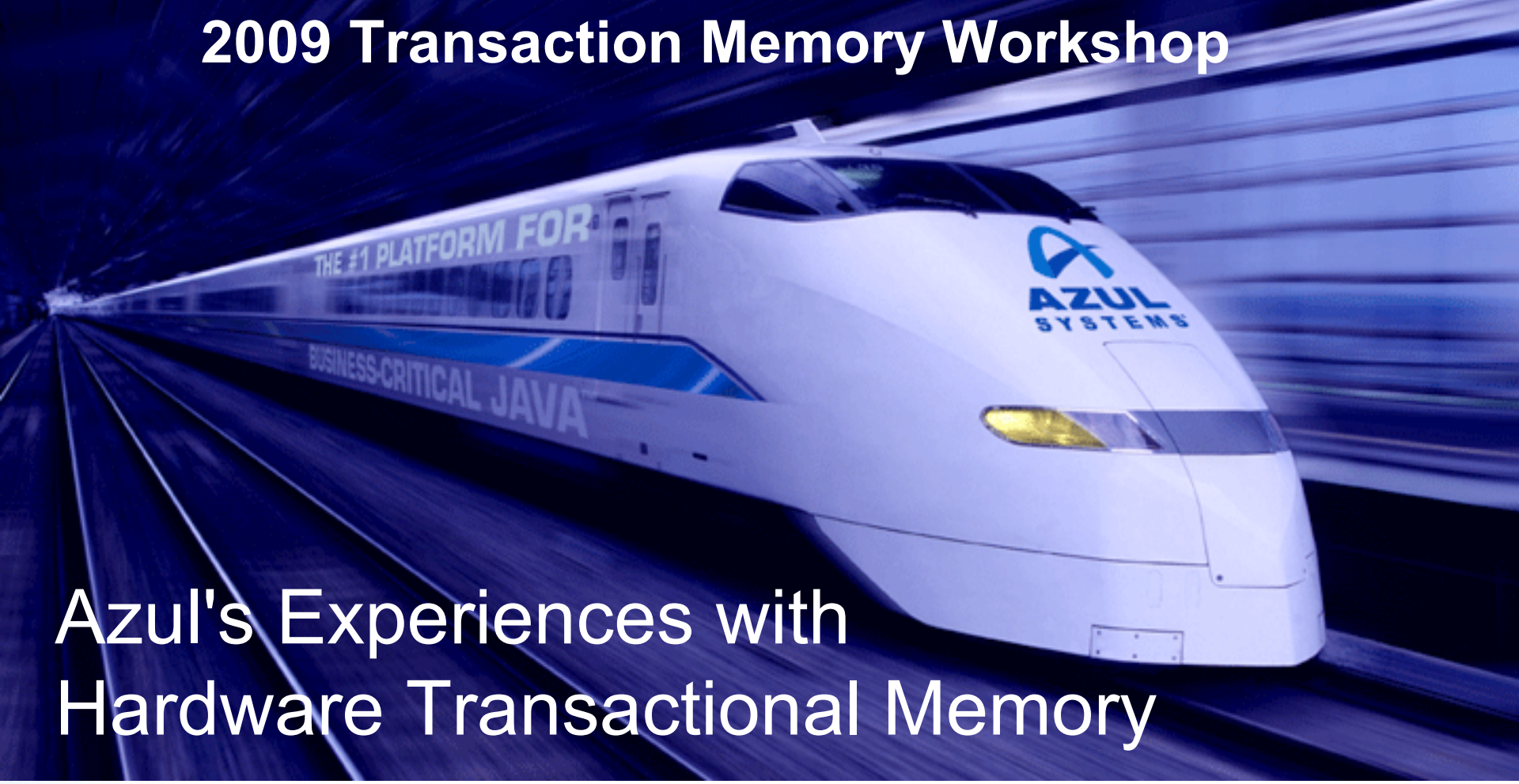
2009 Transaction Memory Workshop



Azul's Experiences with Hardware Transactional Memory

Dr. Cliff Click
Chief JVM Architect & Distinguished Engineer
blogs.azulsystems.com/cliff
Azul Systems
January 31, 2009

AZUL
SYSTEMS

Azul Systems

- Designs our own chips (fab'ed by TSMC)
- Builds our own systems
- Targeted for running business Java
- Large core count - 54 cores per die
  - Up to 16 die are cache-coherent
  - Very weak memory model meets Java spec w/fences
- “UMA” - Flat medium memory speeds
  - Business Java is irregular computation
  - Have supercomputer-level bandwidth
- Modest per-cpu caches
  - 54*(16K+16K) = 1.728Meg fast L1 cache
  - 6*2M = 12M L2 cache
  - Groups of 9 CPUs share L2

Azul Systems

- Cores are classic in-order 64-bit 3-address RISCs
- Each core can sustain 2 cache-missing ops
  - Plus each L2 can sustain 24 prefetches
  - 2300+ outstanding memory references at any time
- Some special ops for Java
  - Read & Write barriers for GC
  - Array addressing and range checks
  - Fast virtual calls
- But core clock rate not real high
- *So task-level parallelism is the name of the game*

The Bottleneck is not the Platform

- JVM scales linear to 864 CPUs
  - Have bandwidth to feed them all as well
- Lite micro-kernel OS
  - Easily supports >100K *runnable* threads
- Heaps >500Gig
  - Sustained allocation rates >40Gig/sec

How do we enable users to write programs with hundreds of runnable threads?

The Bottleneck is not the Platform

- "Big Thread" programs tend to fall into 2 main camps
  - Parallel data "science" (or really "financial modeling") apps
  - Web-tier app-server thread-pool + worklist apps
- Data-parallel apps tend to scale nice
  - After a (short) round of tweaking
  - Although JDK concurrency libs often an issue at > 64 cpus.
- Web-tier apps are more common
  - And scale less well
  - Internal locking of shared structures
  - e.g. legacy uses of Hashtable
- Frequently see <10 cpus without tweaking
- After app tuning see frequently see <50 cpus

Legacy Locking is an Issue

- Lots of Old Java out there
- 3rd-party apps being linked in
  - Source not available (company defunct?)
- Intended to run on 1 cpu (and maybe now 2 cpu) machines
- But lots of the locking is useless
  - ***Data*** contention is rare
  - At least on the *users'* data
  - ***Lock*** contention is far more common
  - e.g. Hashtable
  - But also shared large sync'd HashMaps
  - Lots of other Container classes

No “atomic” keyword

- No desire to enter the “language wars”
- Costumers want old code to “just run faster”
  - Dusty-deck acceleration
- So Azul built TM support to accelerate Java locks
  - Speed up “synchronized” keyword
  - No “atomic” keyword
- Uncontended locks are by far the most common
  - And uncontended locks already fast
  - Azul's CAS and Fences can “hit in cache”
- Contended locks already fast as can be
  - Lite microkernel OS; very fast context switch times

Hardware TM Support

- Built in our first chips
- Leverage L1
  - Extra bits per cache-line
  - “Speculatively-read”, “speculatively-written”
- No changes to L2 *at all*
  - No other CPU is aware that this CPU is attempting a XTN
- SPECULATE instruction
  - Flips CPU speculate mode; starts tracking reads & writes
- ABORT instruction, or abort on eviction from L1
  - Mark spec-lines as “invalid”
- COMMIT instruction
  - Clear spec-bits

Hardware TM Support

- No hardware register rollback
  - Software responsible for all recovery on Abort
  - Possible because only accelerating Java locks
- Limited by size of L1
- Limited by associativity of L1
  - And L2, since shared inclusive L2
- No graceful fallback on Abort
  - No attempt at STM support
- No abort on TLB miss or function call or ...
- Either an XTN fits in cache or it doesn't
- Software heuristics determine when to use the HTM

Software TM Support

- Just “synchronized” keywords
- HotSpot “thin locks” for uncontended locks
  - Just use the fast CAS, no HTM
- Contended locks attempt HTM
  - Success-time/fail-time ratios kept
  - After some initial attempts
    - If ratio is bad, switch to OS lock
  - Periodically re-measure success/fail ratio

Experiences

- Some apps get 2x speedup (e.g. Trade6)
- Most get <10% (e.g. Calypso)
- Lots of teething problems with heuristic
  - Easy to get 10-20% slowdown for constant fail/retry
- Currently turned on always & shipping
- Always see a handful of locks using the HTM
- But rarely the “right” locks to get more CPUs busy

- Failure is almost always to due *conflict* and **not** *capacity*.

XTN Size

- Limited by size & associativity of L1 & L2
- But this appears to be generous:
- XTN's of many thousands of instructions happen
  - Which include 100's of cache-hitting loads
- Most *interesting* XTNs fail for *conflict* and **not** *capacity.*

Issues

- Users don't write “TM friendly” code
- Neither do library writers:
  - e.g. “modcount” - bumped per mod to Hashtable
  - Large shared Hashtable, most updates are unrelated
  - Update itself works well in the HTM
  - But updates to shared “modcount” blow out HTM
  - “modcount” is mostly unused & useless field update
- Same issue with many performance counters
  - Locked writes to a shared variable kills TM
- Many times a small rewrite makes HTM possible
  - But blows the “dusty deck” goal

Issues

- Hard part is to get customer past “dusty deck” thinking
- Once a code rewrite is “on the table”
  - Customer goes whole-hog
  - And rewrites w/fine-grained locking
- Generally only need to “crack” a few locks
- Generally fairly easy, once exact locks are known
- Code then scales on all platforms, not just Azul
- Locks have ***known*** performance issues
  - Better than ***unknown*** TM rollback/retry issues

Hardware Abort is “Too Good”

- Hardware abort rolls back ALL memory writes
  - No “breadcrumbs” on failure
- Hard to record fail reason!
  - Must pass fail code out in a reserved register
  - Even on success path
- First CPUs did not report hardware failure
  - Cannot tell capacity issues from associativity issues from...
- Failure reason is required for a robust heuristic

Summary

- Modest gains
- Rewrites of “dumb” code could help a lot
- Azul did some of this for common JDK pieces
  - Just too much of it Out There
- Future work:
  - Robust customer friendly tool for finding XTN “unfriendly” code
  - Let customer solve the “why does HTM fail?”

http://blogs.azulsystems.com/cliff/

#1 Platform for
Business Critical Java™

WWW.AZULSYSTEMS.COM

THE ERA OF UNBOUND COMPUTE IS NOW

Thank You