

2010 goto;

THE #1 PLATFORM FOR

BUSINESS-CRITICAL JAVA™



# Azul's Experiences with Hardware / Software Co-Design

**Dr. Cliff Click**

Chief JVM Architect & Distinguished Engineer

[blogs.azulsystems.com/cliff](http://blogs.azulsystems.com/cliff)

Azul Systems

Oct 6, 2010



- Design our own chips (fab'ed by TSMC)
- Build our own systems
- Targeted for running business Java
- Large core count - 54 cores per die
  - Up to 16 die are cache-coherent; 864 cores max
  - 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
  - $54 \times (16K + 16K) = 1.728\text{Meg}$  fast L1 cache per die
  - $6 \times 2M = 12M$  L2 cache per die
  - Groups of 9 CPUs share L2

- Cores are classic in-order 64-bit 3-address RISCs
  - Core clock rate lower than X86
- Each core can sustain 2 cache-missing ops
  - Plus each L2 can sustain 24 prefetches
  - 2300+ outstanding memory references at any time
- Hardware Transactional Memory support
- Some special ops for Java
  - Read & Write barriers for GC
  - Array addressing and range checks
  - Fast virtual calls
- *Targeted for thread-level parallelism in managed runtimes*

# 2000-2002 Business Environment



[www.azulsystems.com](http://www.azulsystems.com)

- Java is replacing COBOL (some Y2K driving)
- “App Servers” & J2EE popular – WebSphere, WebLogic, Jboss, “Beans”
- i.e. transactional; task-level parallelism; ThreadPools & Worklists; throughput-oriented computing
- Also CPUs hitting “power wall”
  - Widespread predictions of lower clk freq, more cores
  - ...2010: clk rates stalled @ 3.5GHz but 4-core is commodity
- Obvious synergy: run tasks/transactions on separate cores
- Custom machine to run Java?
  - Who buys custom hardware anymore?
  - Must have **really** good reasons to buy!
  - Mere 5x price/perf not nearly good enough

# What Else Can We Do?



[www.azulsystems.com](http://www.azulsystems.com)

- What else is possible besides pushing “more cores”?
- Big Business Apps require 64-bit heaps
  - Expecting Big Heaps
  - Expecting large thread counts
- GC support; read-barriers in hardware
  - Idea is 20+ yrs old
  - Hardware guys nix 65-bit ptr-tag-in-hardware
    - (65-bit memory requires expensive custom DRAMs)
- Hardware Transactional Memory is “hot” topic
  - And expecting complex task-level parallelism
  - Well understood that complex locking is a problem
  - But nobody wants to rewrite applications w/”atomic”
    - **Still** an open research problem
  - So support for Lock Elision using hybrid software+hardware

# Expect Locking is an Issue



[www.azulsystems.com](http://www.azulsystems.com)

- Uncontended CAS is Fast: most locks are **not** contended
  - (CAS: Compare-And-Swap; unit of atomic update)
- Thin lock is just “CAS + Fence”
  - CAS does **not** memory barrier/fence by default
  - not the right spot for HotSpot & JMM anyways, so HotSpot X86 always fences as well as CAS's
  - CAS can hit-in-cache (1 clk pipelined)
  - Fence can hit-in-cache (1 clk pipelined)
- No-fence CAS
  - several hot use cases: perf counters, lock-free algorithms
- Fence flavors: ld/ld, ld/st, st/ld, st/st
- Not much ordering between mem-ops except for Fence
- Rely on Software (and not e.g. TSO) to get ordering correct

# Expect Locking is an Issue



[www.azulsystems.com](http://www.azulsystems.com)

- HTM Support from Day One
  - speculate & commit (& abort) opcodes
  - Extra tag bits in L1; *nothing* in L2 (hardware guys clear on that!)
- Reads & writes set “spec-read” and “spec-write” tags
- Abort if lose a tagged line out of L1
  - Software recovery; NO hardware register support
- *Nothing else aborts* (contrast to Sun's Rock)
  - i.e. fcn calls OK, TLB miss is OK, nested locks OK
- Routinely see XTNs of 1000's of instructions
  - But not helpful; see other talk
  - Short answer: no dusty-deck speedup from lock-elision
  - And rewriting to break data-dependency allows fine-grained locking
  - And GC is the main bottleneck, not locking

# Expect Bandwidth is an Issue



[www.azulsystems.com](http://www.azulsystems.com)

- Multi-core obvious risk: running out of bandwidth
- Streaming allocation is hard on caches
  - Support for “Just-In-Time” Zero'ing: CLZ
  - That's **not** impacted by frequent fencing for locks (unlike DCBZ)
    - Drove verification guys nuts
  - Lowers bandwidth: no read of dead data
  - Solid 30% reduction in bandwidth
- Stack Allocation support - “Escape Detection”
  - **Much** more effective than Escape Analysis in large programs
    - See IBM results from a few years ago
  - Lowers bandwidth: no write of dead data
- Looser hardware Memory Model than X86/Sparc
  - Rely on JIT to FENCE as needed
  - Makes Scaling to Big Core Counts easier

# Caches & Bandwidth



[www.azulsystems.com](http://www.azulsystems.com)

- Support lots of cache misses (hit-under-miss cores)
  - Similar to Niagra model: want to have lots of slow mem refs active
  - But different from Niagra: full core is as cheap as an SMT core
- Don't really need uber-big caches:
  - goal is *throughput* not single-thread performance
- Lots of memory controllers (4 per chip)
  - Striped memory access ; avoid “hotspots”
  - Successive addresses cycle through all chips
- No fast/local vs slow/remote memory
  - No sane memory layout to allocate “local” vs “remote”
  - 15/16ths of all memory is remote so...
  - ...local access is a loopback off & back on chip
  - Caches work great for stacks & “new” objects
  - Prefetch/CLZ for allocation

# What Else Can We Do?



[www.azulsystems.com](http://www.azulsystems.com)

- Short cache-lines; avoid false-sharing
  - and lowers bandwidth (40% more BW for 64b lines vs 32b lines)
- Faster virtual calls
  - Avoid object header read (cache miss) in common case
  - MetaData already in ptr for GC; might as well do it for v-calls also
- Little stuff:
  - array math & range check ops; sign-extend-then-shift-add
  - IEEE 754 subset
  - fast user-mode traps for all exceptional cases
    - Fast-path hardware, slow-path software
  - Variable-sized register windows – fast function calls
- Cooperative self-suspension
  - expecting to “Safepoint” 1000's of runnable threads

# Eye-Opening Talks w/Hardware Guys



[www.azulsystems.com](http://www.azulsystems.com)

- “I want an instruction that does X!”
  - Reply: “I can give it to you in 3 clks...”
  - ...and here are the 3 1-clk instructions that do X”
- Now show that it's important to do X faster than 3 clks
- In return I got things like:
- “We can directly-execute (most) bytecodes!”
  - Don't bother; it's been tried before
  - JIT'ing is much better; make a nice JIT target instead
- “We can put in a fancy BTB to speed up virtual calls!”
  - Don't bother; software managed inline-caches remove nearly all true virtual calls
- Basic stuff more important to get right

# Core Design Philosophy



[www.azulsystems.com](http://www.azulsystems.com)

- What can we do easier in hardware than in software?
  - e.g. Detection
  - HTM: detecting cache lines *really* hard in software
  - GC Barriers (both read & write)
  - Stack lifetime escape detection
  - Detect inline-cache predicted-virtual-call failure
  - Cache-zero does not order with memory barrier
- What can we do easier in software than in hardware?
  - e.g. all complex fixup logic
  - No register rollback on HTM fail
  - Relocating objects for GC
  - Software Inline-cache vs BTB or other virtual-call support
  - JIT vs direct bytecode execution
  - Fixup for stack-allocated objects escaping stack lifetime

# Expecting OS is an Issue



[www.azulsystems.com](http://www.azulsystems.com)

- No way customer buys funny hardware AND funny OS
- It's a Plug-n-Play Appliance – virtualize the JVM
  - Insert in datacenter network
  - Install new JDK on existing host server
  - 10mins from install to max-score JBB ;:-)
  - No OS for customer to manage
  - No visible compiler tool-chain support; no binary compatibility
  - Speed up older Sun & HP hardware *in-situ*
- Avoid the user-visible OS
  - No device drivers or legacy crud
  - No swap (swap is death for GC)
  - No Big Kernel Lock
  - Existing schedulers not prepared for 100's of CPUs and 1000's of runnable threads

# Expecting OS is an Issue



[www.azulsystems.com](http://www.azulsystems.com)

- Must have hard performance guarantees
  - Move large CPU counts between processes
  - Share unused memory for GC
  - But can demand it back to meet required performance
- Using Virtual Memory extensively for GC
  - Need bulk/fast TLB remapping & shutdown
  - Need bulk/fast virtual-to-physical remapping
  - Want VM anyways for process safety (JVMs DO crash)
- Robustness: ECC caches; chip kill; error reporting; OS de-configure (caches, CPUs & memory chips)
- So roll-our-own “micro” OS

# Why our own CPU?



[www.azulsystems.com](http://www.azulsystems.com)

- Can't find multi-core 64-bit w/ECC CPU design for sale
  - Must redesign L1 & LD/ST unit for HTM, ECC
    - And weak memory model for scaling
  - Adding parity to register file (and later ECC)
  - Meta-data stripping on ld/st
  - Read & write barriers, array ops, v-call support, etc
  - By now redesigning 50% of CPU
- Instruction set non-issue
  - Port gcc + JIT's to any target
  - X86 is nice (high quality ports already; nice tool chain), but only a 'nice'
- So roll our own CPU

# Lots of Cores



[www.azulsystems.com](http://www.azulsystems.com)

- We got lots of CPU cycles
- Anything we can do on another thread is “free”
- Big compiler thread-pools; JIT furiously in background
- Obvious background GC
  - Mutator threads do not trash own cache
  - GC threads on different L2's; trash whole clusters' cache
  - No speed-race for background GC, so running “cacheless” is OK
  - Prefetching in GC is “easy”
- Background profiling, background page zero'ing
- CPUs doing I/O can hot-spin
  - Background CPUs doing scatter/gather, TCP packet work

# Now Design It...



[www.azulsystems.com](http://www.azulsystems.com)

- Hire hardware team
  - Dot-bust puts lots of good engineers on the street
- Hire VM team, hire OS team
- Software team starts porting gcc, HS to new chip AND
- Writing simulator
- Eventually boot OS on simulator AND
- Run HotSpot on fast X86 @ 20Mhz Vega ops
  - Runs SpecJAppserver under simulation
- Lots of cool sim tools built: data-race detector, cache miss rate, cache layout visualizer, trace generation, ...
- **Simulator MUST be run on a *true* multi-cpu machine**
  - **Data-race detection crucial**

# First Cut Design: Vega 1



[www.azulsystems.com](http://www.azulsystems.com)

- 24 cores/chip
  - Grouped in 3 clusters of 8 sharing 1Meg L2 per cluster
- Each core has 16K I & 16K D cache
  - 4-way associative, short 32b line
  - Extra tag bits for HTM
- L2 cluster cache is also 4-way, 32b line
  - Risky for false-sharing of inclusive L1's
  - Limit of die-size & yield
  - Did lots of profiling here
- Clusters full interconnect for 16 chips
  - L2 miss (roughly) same cost to another L2 or to memory
  - No on-chip / off-chip penalty
  - 24 cores/chip x 16 chips = 384 cpus

# First Cut Design



[www.azulsystems.com](http://www.azulsystems.com)

- CPU is easy JIT target
  - Classic in-order 3-adr 32-reg 64-bit RISC
  - 1 hit-under-miss cache; 1-entry store “latch”
  - Masking of metadata in ptrs on loads & stores
  - Very simple FPU; no FPRs; no flags; no modes
- Background spill/fill for register stack
- Special ops almost all do minor ALU op & fast user trap:
  - array math & range check; replaces 2-5 ops each
  - V-call avoids a cache-missing load; replaces 3-4 ops
  - Read barrier: also includes TLB probe
  - Write barrier: replaces 20+ integer ops
    - But only because doing complex Stack-Escape barrier
    - Card-Mark-only generational GC would replace only 3-5 ops

# Two years later (2004)...



[www.azulsystems.com](http://www.azulsystems.com)

- First silicon comes back from TSMC
  - Not quite Dead On Arrival
  - L2 death kills most clusters
  - But a few L2's can run w/ECC & 1-way “limp home” mode
  - Register writes from even-registers “bleed” into odd registers
    - So only JIT to a subset of registers
    - Decoder treats branch offset bits as registers
    - So only branch to even addresses, etc
  - Still get a few “good” chips; must over-voltage them to make registers behave so chips are “cooked” to death in a month...
- *So SW makes progress while HW fixes chip!*
- 2<sup>nd</sup> silicon; metal-mask spin only
  - Mostly functional
- 3<sup>rd</sup> silicon: metal-mask spin only; crucial security bug-fix

# Two years later (2004)...



- Two weeks from silicon arriving to booting OS
  - One day later “hello, world”
  - Four days later “java -version”
  - All those simulator hours REALLY paid off
- Still took a year to get system robust
  - Not just metal-spins – true data-race bug fixes
    - Nobody's seen a system this O-O-O & concurrent before
  - Performance warts
    - That 4-way inclusive L2 causing endless conflicts
    - Also heavy TLB misses
    - Random offset stacks & JIT code-cache & page coloring fixes it
  - Turns out 4-way L2 sharing 8 4-way L1's IS ok
  - Virtualization layer not virtual enough
  - And not performant enough
    - Lots of software performance fixes through the years

# First customer!



[www.azulsystems.com](http://www.azulsystems.com)

- 2004 May - 1<sup>st</sup> silicon
- 2005 Jan - 1<sup>st</sup> Beta – *this is amazingly fast!!!!*
- 2005 June - 1<sup>st</sup> paying customer, Pegasus Systems doing hotel booking
- 2005 Nov - Then British Telecom doing B2B
- Then 2006 Credit Suisse, then another big bank, then another, ...

# What Works, What Doesn't



[www.azulsystems.com](http://www.azulsystems.com)

- Chip works (after 2<sup>nd</sup> metal spin)
  - Plenty of bandwidth & CPU cycles
  - Predicted cache miss rates (eventually) achieved
  - Still CPUs slower than hoped for
    - Limit of in-order low-frequency core
    - First CPU has only 1 outstanding miss
  - And many new hardware features not “turned on”
- Software works
  - Stability is 1<sup>st</sup> priority
  - So new hardware features not enabled for quite some time
  - VM team has hands full w/basic code-gen JIT quality & dataraces

# What Works, What Doesn't



[www.azulsystems.com](http://www.azulsystems.com)

- Hardware has teething problems for a year
  - Weird low-frequency DRAM bugs
    - Many issues masked by ECC
    - Forces OS error reporting to become robust early
  - DRAM screening a nightmare: need 128 **good** DIMMs
  - Can't get a power supply that's as reliable as claimed
  - Motherboard & I/O ASIC goes through several iterations
- OS – teething problems
  - The scheduler goes through several rounds
  - So does the I/O stack
    - **Efficient** virtualization is hard
- VM – read barriers
  - Must have read barriers everywhere
  - Every integration from Sun brings in new un-barriered loads
  - GC churns rapidly; exposes unprotected OOPs in VM code

# New Feature Issues



[www.azulsystems.com](http://www.azulsystems.com)

- Engineering priority debate rages over:
  - Stability
  - Turning on HTM, stack allocation (e.g. new chip features)
  - Vs compiler thread pools (startup time), tiered compilation (faster single-thread performance)
  - Vs generational GC (efficiency)
  - Vs GC pause-time improvements (e.g. concurrent SystemDictionary updates)
  - Vs fixing JDK scaling warts
  - Vs improving internal VM tool chain
- Both HTM & Stack Allocation lose for awhile
  - Lack of engineering man hours; hard problems
  - Engineers e.g. helping with sales calls
  - Customers seeing true data-races in their buggy code

# Eventually HTM Turned On



[www.azulsystems.com](http://www.azulsystems.com)

- HTM performance buggy for quite awhile
  - Mostly in “live lock”: endless retry/fail loops
  - Need to fail to OS sooner, but also retry HTM again periodically
- Turned on by default & shipping for 4 yrs now
  - Rarely helps customers; (almost) never hurts
- Stack Allocation has more issues
  - Standard case is really good:
    - 70% of all objects in a big busy app-server get stack allocated
  - Bad cases are really bad – endless stack escapes
  - And our standard GC is also really good
  - So no drive to fix bad cases
  - Not turned on by default

# What Works



[www.azulsystems.com](http://www.azulsystems.com)

- GC works *really* well now
  - No sweat handling 500G heaps
  - Or 35G/sec allocation rates
- First time at a new customer
  - (1) Install
  - (2) Strip all old GC args; double default heap size
  - (3) Run – no GC problems (ever again)
- Showed off internal profiling VM tool “RTPM”
  - Customers demanded it
  - Now major selling point
- Chips, OS solid
  - Uptimes of over a year on many systems
  - Most downtimes now caused by e.g. datacenter cooling failures (e.g. nothing to do w/Azul)

# Real Time Profiling & Monitoring



[www.azulsystems.com](http://www.azulsystems.com)

- #2 feature (behind GC & stable performance under load)
- Live peek into JVM guts w/any web browser
- Always on, no overhead, monitoring
- Live thread stacks
- Hot Locks & blocking backtraces
- Live & Allocated Heap objects; leak detection
- GC speeds & feeds; I/O speeds & feeds; file cache
- Hot ticks; JIT'd code w/ticks
- Error reporting & exceptional conditions

# Rolling Along...



[www.azulsystems.com](http://www.azulsystems.com)

- 2006: Vega 2: 48 cpus/chip; higher clock; faster mem bus
  - Java 1.5 JVM
  - Tweaks to Read Barrier HW to support generational GC
  - Drop some less used instructions (not binary compatible)
- 2008: shipping Vega 3: 54 cores/chip; 2Meg L2; higher clk
  - Java 1.6 JVM
  - Generational GC
  - Better profiling support
  - Better HTM reporting
- Now working on 4<sup>th</sup> gen

# Some Lessons Learned



www.azulsystems.com

- Owning whole stack allows progress:
  - JVM, OS can work around really bad HW bugs
  - Some HW bugs “fixed” forever in SW
- Some really hard HW problems “solved” in SW
  - CLZ cuts bandwidth by 1/3
- GC is “solved” w/HW Read Barrier
  - Or at least we can handle 500G heaps & 35G/sec allocation rates
  - With max pause of 10-20ms
- Simple HTM **can** do Lock Elision
  - But it doesn't really help scalability
  - Might help N-CAS algorithms in libraries
- Huge count of simple cores really useful in production

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

***#1 Platform for  
Business Critical Java™***

**WWW.AZULSYSTEMS.COM**

.....THE ERA OF UNBOUND COMPUTE IS NOW.....

Thank You

