

CPU Caches

Jamie Allen

@jamie_allen

<http://github.com/jamie-allen>

JavaZone 2012

Agenda

- Goal
- Definitions
- Architectures
- Development Tips
- The Future

Goal

Provide you with the information you need
about CPU caches so that you can improve the
performance of your applications

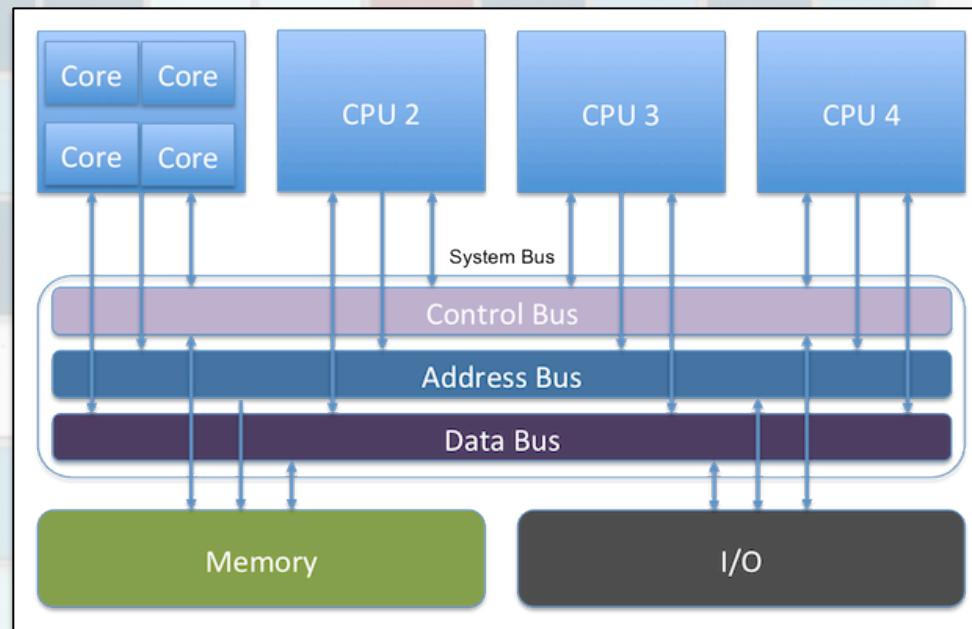
Why?

- Increased virtualization
 - Runtime (JVM, RVM)
 - Platforms/Environments (cloud)
- Disruptor, 2011

Definitions

SMP

- Symmetric Multiprocessor (SMP) Architecture
- Shared main memory controlled by single OS
- No more Northbridge



Typesafe

NUMA

- Non-Uniform Memory Access
- The organization of processors reflect the time to access data in RAM, called the “NUMA factor”
- Shared memory space (as opposed to multiple commodity machines)

Data Locality

- The most critical factor in performance
- Not guaranteed by a JVM
- **Spatial** - reused over and over in a loop, data accessed in small regions
- **Temporal** - high probability it will be reused before long

Memory Controller

- Manages communication of reads/writes between the CPU and RAM
- Integrated Memory Controller on die

Cache Lines

- 32-256 contiguous bytes, most commonly 64
- Beware “false sharing”
- Use padding to ensure unshared lines
- Transferred in 64-bit blocks (8x for 64 byte lines), arriving every ~4 cycles
- Position in the line of the “critical word” matters, but not if pre-fetched
- `@Contended` annotation coming to JVM?

Cache Associativity

- **Fully Associative:** Put it anywhere
- **Somewhere in the middle:** n-way set-associative, 2-way skewed-associative
- **Direct Mapped:** Each entry can only go in one specific place

Cache Eviction Strategies

- Least Recently Used (LRU)
- Pseudo-LRU (PLRU): for large associativity caches
- 2-Way Set Associative
- Direct Mapped
- Others

Cache Write Strategies

- **Write through:** changed cache line immediately goes back to main memory
- **Write back:** cache line is marked when dirty, eviction sends back to main memory
- **Write combining:** grouped writes of cache lines back to main memory
- **Uncacheable:** dynamic values that can change without warning

Exclusive versus Inclusive

- Only relevant below L3
- AMD is exclusive
 - Progressively more costly due to eviction
 - Can hold more data
 - Bulldozer uses "write through" from L1d back to L2
- Intel is inclusive
 - Can be better for inter-processor memory sharing
 - More expensive as lines in L1 are also in L2 & L3
 - If evicted in a higher level cache, must be evicted below as well

Inter-Socket Communication

- GT/s – gigatransfers per second
- Quick Path Interconnect (QPI, Intel) – 8GT/s
- HyperTransport (HTX, AMD) – 6.4GT/s (?)
- Both transfer 16 bits per transmission in practice, but Sandy Bridge is really 32

MESI+F Cache Coherency Protocol

- Specific to data cache lines
- **Request for Ownership (RFO)**, when a processor tries to write to a cache line
- **Modified**, the local processor has changed the cache line, implies only one who has it
- **Exclusive**, one processor is using the cache line, not modified
- **Shared**, multiple processors are using the cache line, not modified
- **Invalid**, the cache line is invalid, must be re-fetched
- **Forward**, designate to respond to requests for a cache line
- All processors **MUST** acknowledge a message for it to be valid

Static RAM (SRAM)

- Requires 6-8 pieces of circuitry per datum
- Cycle rate access, not quite measurable in time
- Uses a relatively large amount of power for what it does
- Data does not fade or leak, does not need to be refreshed/recharged

Dynamic RAM (DRAM)

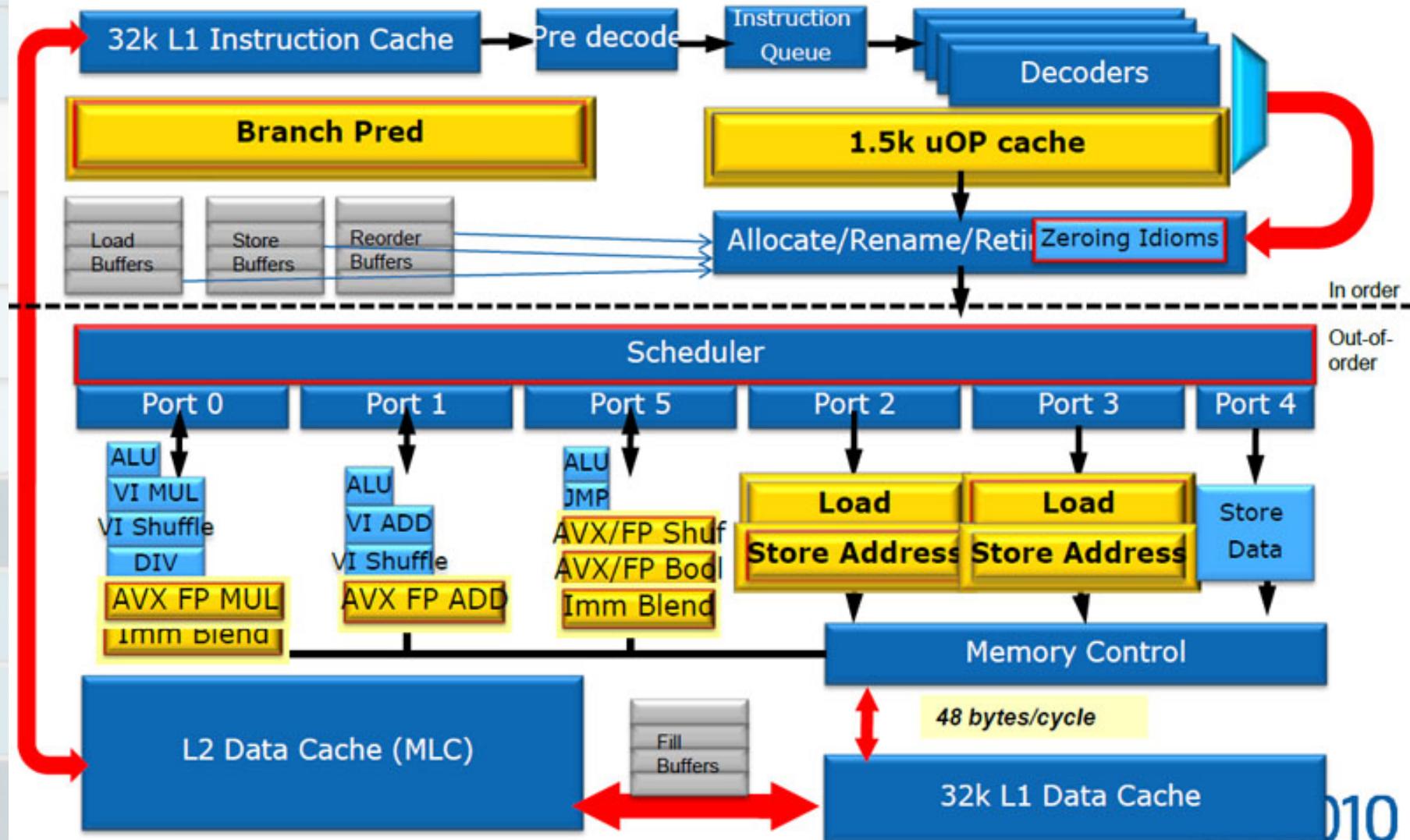
- Requires 2 pieces of circuitry per datum
- “Leaks” charge, but not sooner than 64ms
- Reads deplete the charge, requiring subsequent recharge
- Takes 240 cycles (~100ns) to access
- Intel's Nehalem architecture - each CPU socket controls a portion of RAM, no other socket has direct access to it

Architectures

Current Processors

- **Intel**
 - Nehalem/Westmere
 - Sandy Bridge
 - Ivy Bridge
- **AMD**
 - Bulldozer
- **Oracle**
 - UltraSPARC isn't dead

Sandy Bridge Microarchitecture



“Latency Numbers Everyone Should Know”

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns = 3 µs
Send 2K bytes over 1 Gbps network	20,000 ns = 20 µs
SSD random read	150,000 ns = 150 µs
Read 1 MB sequentially from memory	250,000 ns = 250 µs
Round trip within same datacenter	500,000 ns = 0.5 ms
Read 1 MB sequentially from SSD*	1,000,000 ns = 1 ms
Disk seek	10,000,000 ns = 10 ms
Read 1 MB sequentially from disk	20,000,000 ns = 20 ms
Send packet CA->Netherlands->CA	150,000,000 ns = 150 ms

- Shamelessly cribbed from this gist: <https://gist.github.com/2843375>, originally by Peter Norvig and amended by Jeff Dean

Measured Cache Latencies

Sandy Bridge-E	L1d	L2	L3	Main
<hr/>				
Sequential Access	3 clk	11 clk	14 clk	6ns
Full Random Access	3 clk	11 clk	38 clk	65.8ns

SI Software's benchmarks: http://wwwsisoftware.net/?d=qa&f=ben_mem_latency

Registers

- On-core for instructions being executed and their operands
- Can be accessed in a single cycle
- There are many different types
- A 64-bit Intel Nehalem CPU had 128 Integer & 128 floating point registers

Store Buffers

- Hold data for Out of Order (OoO) execution
- Fully associative
- Prevent “stalls” in execution on a thread when the cache line is not local to a core on a write
- ~1 cycle

Level Zero (L0)

- New to Sandy Bridge
- A cache of the last 1536 uops decoded
- Well-suited for hot loops
- Not the same as the older "trace" cache

Level One (L1)

- Divided into data and instructions
- 32K data (L1d), 32K instructions (L1i) per core on a Sandy Bridge
- Sandy Bridge loads data at 256 bits per cycle, double that of Nehalem
- 3-4 cycles to access L1d

Level Two (L2)

- 256K per core on a Sandy Bridge
- 2MB per “module” on AMD's Bulldozer architecture
- ~11 cycles to access
- Unified data and instruction caches from here up
- If the working set size is larger than L2, misses grow

Level Three (L3)

- Was a “unified” cache up until Sandy Bridge, shared between cores
- Varies in size with different processors and versions of an architecture
- 14-38 cycles to access

Programming Tips

Striding & Pre-fetching

- Predictable memory access is really important
- Hardware pre-fetcher on the core looks for patterns of memory access
- Can be counter-productive if the access pattern is not predictable
- Martin Thompson blog post:
[“Memory Access Patterns are Important”](#)
- Shows the importance of locality and striding

Cache Misses

- Cost hundreds of cycles
- Keep your code simple
- Instruction read misses are most expensive
- Data read miss are less so, but still hurt performance
- Write misses are okay unless using Write Through
- Miss types:
 - Compulsory
 - Capacity
 - Conflict

Programming Optimizations

- Stack allocated data is cheap
- Pointer interaction - you have to retrieve data being pointed to, even in registers
- Avoid locking and resultant kernel arbitration
- CAS is better and occurs on-thread, but algorithms become more complex
- Match workload to the size of the last level cache (LLC, L3)

What about Functional Programming?

- Have to allocate more and more space for your data structures, leads to eviction
- When you cycle back around, you get cache misses
- Choose immutability by default, profile to find poor performance
- Use mutable data in targeted locations

Hyperthreading

- Great for I/O-bound applications
- If you have lots of cache misses
- Doesn't do much for CPU-bound applications
- You have half of the cache resources per core

Data Structures

- **BAD:** Linked list structures and tree structures
- **BAD:** Java's HashMap uses chained buckets
- **BAD:** Standard Java collections generate lots of garbage
- **GOOD:** Array-based and contiguous in memory is much faster
- **GOOD:** Write your own that are lock-free and contiguous
- **GOOD:** Fastutil library, but note that it's additive

Application Memory Wall & GC

- Tremendous amounts of RAM at low cost
- GC will kill you with compaction
- Use pauseless GC
 - IBM's Metronome, very predictable
 - Azul's C4, very performant

Using GPUs

- Remember, locality matters!
- Need to be able to export a task with data that does not need to update

The Future

ManyCore

- David Ungar says > 24 cores, generally many 10s of cores
- Really gets interesting above 1000 cores
- Cache coherency won't be possible
- Non-deterministic

Memristor

- Non-volatile, static RAM, same write endurance as Flash
- 200-300 MB on chip
- Sub-nanosecond writes
- Able to perform processing?
- Multistate, not binary

Phase Change Memory (PRAM)

- Higher performance than today's DRAM
- Intel seems more fascinated by this, just released its "neuromorphic" chip design
- Not able to perform processing
- Write degradation is supposedly much slower
- Was considered susceptible to unintentional change, maybe fixed?

Thanks!



Typesafe

Credits!

- [What Every Programmer Should Know About Memory](#), Ulrich Drepper of RedHat, 2007
- [Java Performance](#), Charlie Hunt
- [Wikipedia/Wikimedia Commons](#)
- [AnandTech](#)
- [The Microarchitecture of AMD, Intel and VIA CPUs](#)
- [Everything You Need to Know about the Quick Path Interconnect](#), Gabriel Torres/Hardware Secrets
- [Inside the Sandy Bridge Architecture](#), Gabriel Torres/Hardware Secrets
- Martin Thompson's [Mechanical Sympathy blog](#) and Disruptor presentations
- [The Application Memory Wall](#), Gil Tene, CTO of Azul Systems
- [AMD Bulldozer/Intel Sandy Bridge Comparison](#), Gionatan Danti
- [SI Software's Memory Latency Benchmarks](#)
- Martin Thompson and Cliff Click provided feedback &additional content