Computer Science 703

Advance Computer Architecture

2010 Semester 1

Lecture Notes 9

19Mar10

Memory Systems

James Goodman

Department of Computer Science

2010 YEAR PRESENTATION The University of Auckland | New Zealand

Coming Lectures

- Next week (week 4)
  - Programming Parallel Systems
- Week 5
  - Parallel computing with shared memory

2010 YEAR PRESENTATION The University of Auckland | New Zealand

How to Survive the Multicore Software Revolution (or at Least Survive the Hype)
http://software.intel.com/en-us/articles/e-book-on-multicore-programming/
http://software.intel.com/file/23369

OpenMP Tutorial:
https://computing.llnl.gov/tutorials/openMP/

Pthreads Tutorials:
https://computing.llnl.gov/tutorials/pthreads/
http://cs.gmu.edu/~white/CS571/pthreadTutorial.pdf

MPI Tutorial:
https://computing.llnl.gov/tutorials/mpi/

Topics for revision: The C Programming Language -- Kernighan and Ritchie (Amazon, Google, Library)

Processes and Threads
http://en.wikipedia.org/wiki/Process_(computing)
http://en.wikipedia.org/wiki/Thread_(computer_science)

Additional Reading:

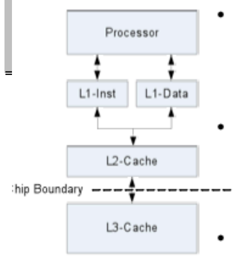
Ars Technica Snow Leopard Review (pages 11-12)
http://arstechnica.com/apple/reviews/2009/08/mac-os-x-10-6.ars/11
http://arstechnica.com/apple/reviews/2009/08/mac-os-x-10-6.ars/12

2010 YEAR PRESENTATION The University of Auckland | New Zealand

Memory Systems (Caches)

- Basic Ideas & Organization
  - Placement: hashing
  - Replacement: LRU; approximate LRU; random
  - Block size
  - Associativity
  - Capturing temporal/spatial locality
  - Writing
    - write-through/write-back
    - write-allocation
    - write buffers
  - Capacity, Compulsory & Conflict misses
  - Hit/Miss ratio; Avg. Memory Access Time (AMAT)
- Cache hierarchy
  - Registers, cache, main memory, disk
  - **Inclusion vs. exclusion**
  - Optimizations
  - Specialized (I-cache, D-cache, TLB)
  - Victim cache
- **Requirements of main memory**
  - **Blocking/multiple requests**
  - Interleaving; "false interleaving"
  - **Miss Holding Status Registers MSHR**
- I/O and other conflicts
- Virtual memory (latency and aliasing)

Multilevel Caches



- Motivation:
  - Optimize each cache for different constraints
  - Exploit cost/capacity trade-offs at different levels
- L1 caches
  - Optimized for fast access time (1-3 CPU cycles)
  - 8KB-64KB, DM to 4-way SA
- L2 caches
  - Optimized for low miss rate (off-chip latency high)
  - 256KB-4MB, 4- to 16-way SA
- L3 caches
  - Optimized for low miss rate (DRAM latency high)
  - Multi-MB, highly associative, embedded DRAM?

2-level Cache Performance Equations

- L1 AMAT = HitTimeL1 + MissRateL1 * Miss PenaltyL1
  - MissLatencyL1 is low, so optimize HitTimeL1
- MissPenaltyL1 = HitTimeL2 + MissRateL2 * MissPenaltyL2
  - MissLatencyL2 is high, so optimize MissRateL2
- MissPenaltyL2 = DRAMaccessTime + (BlockSize/Bandwidth)
  - If DRAM time high or bandwidth high, use larger block size

- L2 miss rate:
  - Global: L2 misses / total CPU references
  - Local: L2 misses / CPU references that miss in L1
  - The equation above assumes local miss rate

Multi-level Inclusion

- Inclusion: if data at L1 is <u>always a subset</u> of data at L2

- Advantages of maintaining multi-level inclusion
  - Easier cache analysis
    - Overall MissRate = MissRate_{L1} x $\text{LocalMissRate}_{L2}$
  - Easier coherence checks for I/O & multiprocessors
    - Check the lowest level only to determine if data in cache

- Disadvantages
  - L2 replacements are complicated if L2 and L1 block sizes differ
  - Wasted space if L2 not much larger than L1
    - The motivation for non-inclusion for some AMD chips

How to Maintain Inclusion

- On L1 misses
  - Bring block in L2 as well

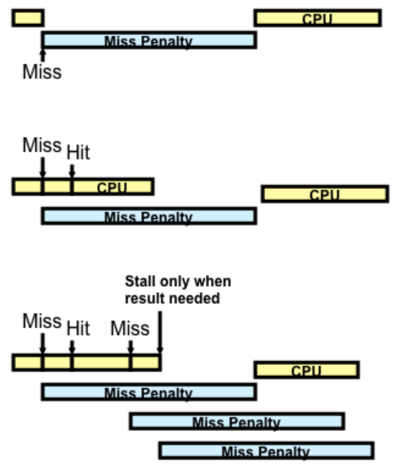
- On L2 evictions or invalidations
  - First evict all block(s) from L1
  - Can simplify by maintaining extra state in L2 indicates which blocks are also in L1 and where (cache way)

- L1 instruction cache inclusion?
  - For most systems, instruction inclusion is not needed (why?)
  - Bad for applications that stress the L2 capacity with small code
    - E.g. matrix multiply with huge matrices…

Non-blocking or Lockup Free Caches

- Idea:
  - Allow for hits while serving a miss (hit-under-miss)
  - Allow for more than one outstanding miss (miss-under-miss)
- When does it make sense (for L1, L2, …)
  - When the processor can handle >1 pending load/store
    - This is the case with superscalar processors
  - When the cache serves >1 processor or other cache
  - When the lower level allows for multiple pending accesses
    - Multi-banked, split transaction busses, pipelining, …
- What is difficult about non-blocking caches:
  - Handling multiple misses at the time
  - Handling loads to pending misses
  - Handling stores to pending misses

Potential of Non-blocking Caches



Stall CPU on miss

Hit under miss

Multiple out-standing misses

Miss Status Handling Register

- Keeps track of
  - Outstanding cache misses
  - Pending load & stores that refer to that cache block
- Fields of an MSHR
  - Valid bit
  - Cache block address
    - Must support associative search
  - Issued bit (1 if already request issued to memory)
  - For each pending load or store
    - Valid bit
    - Type (load/store) and format (byte/halfword/…)
    - Block offset
    - Destination register for load OR store buffer entry for stores

MSHR

1	27	1
Valid	Block Address	Issued

1	3	5	5	
Valid	Type	Block Offset	Destination	Load/store 0
Valid	Type	Block Offset	Destination	Load/store 1
Valid	Type	Block Offset	Destination	Load/store 2
Valid	Type	Block Offset	Destination	Load/store 3