CS 61C: Great Ideas in Computer Architecture (Machine Structures)

Instructors:
Randy H. Katz
David A. Patterson
http://inst.eecs.Berkeley.edu/~cs61c/fa10

Agenda

- Cache Sizing/Hits and Misses
- Administrivia
- Technology Break
- Cache Performance

Agenda

- Cache Sizing/Hits and Misses
- Administrivia
- Technology Break
- Cache Performance

Cache Field Sizes

- Number of bits in a cache includes both the storage for data and for the tags
  - 32-bit byte address
  - For a direct mapped cache with 2^n blocks, n bits are used for the index
  - For a block size of 2^m words (2^{m+2} bytes), m bits are used to address the word within the block and 2 bits are used to address the byte within the word
- What is the size of the tag field?
- Total number of bits in a direct-mapped cache is then
  - 2^n x (block size + tag field size + valid field size)

Peer Instruction

- How many total bits are required for a direct mapped cache with 16KB of data and 4-word blocks assuming a 32-bit address?
  - A: 128K bits
  - B: 19K bits
  - C: 147K bits
  - D: 148K bits
  - E: 129K bits
  - F: 18K bits

Handling Cache Hits

- Read hits (I$ and D$)
  - Hits are good in helping us go fast
  - Misses are bad/slow us down
- Write hits (D$ only)
  - Require cache and memory to be consistent
    - Write-through: Always write the data into the cache block and the next level in the memory hierarchy
    - Writes run at the speed of next level in memory hierarchy – so slow! – or can use a write buffer and stall only if the write buffer is full
  - Allow cache and memory to be inconsistent
    - Write-back: Write the data only into the cache block (cache block written back to next level in memory hierarchy when it is "evicted")
    - Need a dirty bit for each data cache block to tell if it needs to be written back to memory when evicted – can use a write buffer to help "buffer" write-backs of dirty blocks

Sources of Cache Misses

- *Compulsory* (cold start or process migration, first reference):
  - First access to a block, "cold" fact of life, not a whole lot you can do about it. If you are going to run "millions" of instruction, compulsory misses are insignificant
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- *Capacity*:
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size (may increase access time)
- *Conflict* (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity (stay tuned!) (may increase access time)

Handling Cache Misses (Single Word Blocks)

- Read misses (I$ and D$)
  - Stall the pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send requested word to processor, then let pipeline resume
- Write misses (D$ only)
  - Stall the pipeline, fetch the block from next level in the memory hierarchy, install it in cache (may involve evicting a dirty block if using write-back), write the word from processor to cache, then let pipeline resume

  or
  - Write allocate: just write word into the cache updating both tag and data; no need to check for cache hit, no need to stall

  or
  - No-write allocate: skip the cache write (but must invalidate cache block since it will now hold stale data) and just write the word to write buffer (and eventually to the next memory level); no need to stall if write buffer isn't full

Handling Cache Misses (Multiword Block Considerations)

- Read misses (I$ and D$)
  - Processed the same as for single word blocks – a miss returns the entire block from memory
  - Miss penalty grows as block size grows
    - Early restart: processor resumes execution as soon as the requested word of the block is returned
    - Requested word first: requested word is transferred from the memory to the cache (and processor) first
  - Nonblocking cache – allows the processor to continue to access cache while cache is handling an earlier miss
- Write misses (D$)
  - If using write allocate must first fetch block from memory and then write word to block (or could end up with a "garbled" block in the cache.
  - E.g., for 4 word blocks, a new tag, one word of data from the new block, and three words of data from the old block)

Agenda

- Cache Sizing/Hits and Misses
- Administrivia
- Technology Break
- Cache Performance

Midterm!

- TA Exam Review Tonight!, 100 GPB 6-8 PM
- NO lecture next Wednesday, 6 October
- Exam, 6-9 PM, 1 Pimentel
- Everything covered through today
- Key topics:
  - C programming (pointers, arrays, structures)
  - Mapping C into assembly/MIPS instructions
  - General technology trends
  - Components of a computer
  - Request and data level parallelism
  - Quantitative performance and benchmarking
  - Memory Hierarchy/Caches

Agenda

- Cache Sizing/Hits or Misses
- Administrivia
- Technology Break
- Cache Performance

Agenda

- Cache Sizing/Hits and Misses
- Administrivia
- Technology Break
- Cache Performance

Measuring Cache Performance

- Assuming cache hit costs are included as part of the normal CPU execution cycle, then

  CPU time = IC × CPI × CC

  = IC × (CPI_{ideal} + Memory-stall cycles) × CC

  (CPI_{stall})

- Memory-stall cycles come from cache misses (a sum of read-stalls and write-stalls)

  Read-stall cycles = reads/program × read miss rate × read miss penalty

  Write-stall cycles = (writes/program × write miss rate × write miss penalty) + write buffer stalls

- For write-through caches, we can simplify this to

  Memory-stall cycles = accesses/program × miss rate × miss penalty

Impacts of Cache Performance

- Relative $ penalty increases as processor performance improves (faster clock rate and/or lower CPI)
  - Memory speed unlikely to improve as fast as processor cycle time. When calculating CPI_{stall}, cache miss penalty is measured in processor clock cycles needed to handle a miss
  - Lower the CPI_{ideal}, more pronounced impact of stalls
- Processor with a CPI_{ideal} of 2, a 100 cycle miss penalty, 36% load/store instr's, and 2% I$ and 4% D$ miss rates
  - Memory-stall cycles = 2% × 100 + 36% × 4% × 100 = 3.44
  - So CPI_{stalls} = 2 + 3.44 = 5.44
  - More than twice the CPIideal !
- What if the CPI_{ideal} is reduced to 1? 0.5? 0.25?
- What if the D$ miss rate went up by 1%? 2%?
- What if the processor clock rate is doubled/cycle halved (miss penalty doubled)?

Average Memory Access Time (AMAT)

- Larger $ has longer access time. Increase in hit time will likely add another stage to the pipeline. At some point, increase in hit time for a larger cache will overcome the improvement in hit rate, yielding a decrease in performance.
- Average Memory Access Time (AMAT) is the average to access memory considering both hits and misses

AMAT = Time for a hit + Miss rate x Miss penalty

- What is the AMAT for a processor with a 20 psec clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache access time of 1 clock cycle?

1 + 0.02 x 50 = 2

Reducing Cache Miss Rates

- Use multiple $ levels
- With advancing technology, have more room on die for bigger L1 caches or for second level cache – normally a unified L2 cache (i.e., it holds both instructions and data,) and in some cases even a unified L3 cache
- E.g., CPI_{ideal} of 2,
  100 cycle miss penalty (to main memory),
  25 cycle miss penalty (to UL2$),
  36% load/stores,
  a 2% (4%) L1 I$ (D$) miss rate,
  add a 0.5% UL2$ miss rate
  - CPI_{stalls} = 2 + .02×25 + .36×.04×25 + .005×100 + .36×.005×100
    = 3.54 (vs. 5.44 with no L2$)

Multilevel Cache Design Considerations

- Different design considerations for L1$ and L2$
  - Primary $ focuses on minimizing hit time for shorter clock cycle: Smaller $ with smaller block sizes
  - Secondary $(s) focus on reducing miss rate to reduce penalty of long main memory access times: Larger $ with larger block sizes/higher levels of associativity
- Miss penalty of L1$ is significantly reduced by presence of L2$, so can be smaller/faster but with higher miss rate
- For the L2$, hit time is less important than miss rate
  - L2$ hit time determines L1$'s miss penalty
  - L2$ local miss rate >> than the global miss rate

Improving Cache Performance (1 of 3)

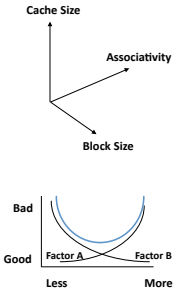
0. Reduce the time to hit in the cache
   - Smaller cache
   - Direct mapped cache
   - Smaller blocks
   - For writes
     - No write allocate – no "hit" on cache, just write to write buffer
     - Write allocate – to avoid two cycles (first check for hit, then write) pipeline writes via a delayed write buffer to cache
1. Reduce the miss rate
   - Bigger cache
   - More flexible placement (increase associativity)
   - Larger blocks (16 to 64 bytes typical)
   - Victim cache – small buffer holding most recently discarded blocks

Improving Cache Performance (2 of 3)

2. Reduce the miss penalty
   - Smaller blocks
   - Use a write buffer to hold dirty blocks being replaced so don't have to wait for the write to complete before reading
   - Check write buffer (and/or victim cache) on read miss – may get lucky
   - For large blocks fetch critical word first
   - Use multiple cache levels – L2 cache not tied to CPU clock rate
   - Faster backing store/improved memory bandwidth
     - Wider buses
     - Memory interleaving, DDR SDRAMs

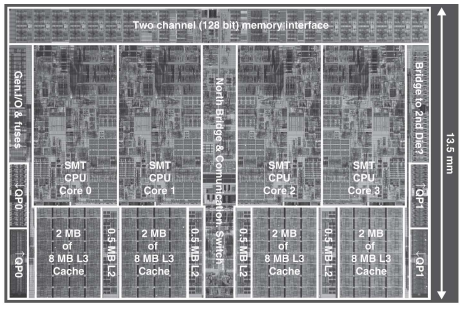
The Cache Design Space (3 of 3)

- Several interacting dimensions
  - Cache size
  - Block size
  - (Associativity)
  - Replacement policy
  - Write-through vs. write-back
  - Write allocation
- Optimal choice is a compromise
  - Depends on access characteristics
    - Workload
    - Use (I-cache, D-cache, TLB)
  - Depends on technology / cost
- Simplicity often wins



Characteristic	Intel Nehalem	AMD Opteron X4 (Barcelona)
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KB each for instructions/data per core	64 KB each for instructions/data per core
L1 cache associativity	4-way (I), 8-way (D) set associative	2-way set associative
L1 replacement	Approximated LRU replacement	LRU replacement
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L1 hit time (load-use)	Not Available	3 clock cycles
L2 cache organization	Unified (instruction and data) per core	Unified (instruction and data) per core
L2 cache size	256 KB (0.25 MB)	512 KB (0.5 MB)
L2 cache associativity	8-way set associative	16-way set associative
L2 replacement	Approximated LRU replacement	Approximated LRU replacement
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	Not Available	9 clock cycles
L3 cache organization	Unified (instruction and data)	Unified (instruction and data)
L3 cache size	8192 KB (8 MB), shared	2048 KB (2 MB), shared
L3 cache associativity	16-way set associative	32-way set associative
L3 replacement	Not Available	Evict block shared by fewest cores
L3 block size	64 bytes	64 bytes
L3 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L3 hit time	Not Available	38 (?)clock cycles

Intel Nehalem Chip Photo



CPI/Miss Rates/DRAM Access SpecInt2006

Name	CPI	L1 D cache misses/1000 instr	L2 D cache misses/1000 instr	DRAM accesses/1000 instr
perl	0.75	3.5	1.1	1.3
bzip2	0.85	11.0	5.8	2.5
gcc	1.72	24.3	13.4	14.8
mcf	10.00	106.8	88.0	88.5
go	1.09	4.5	1.4	1.7
hmmer	0.80	4.4	2.5	0.6
sjeng	0.96	1.9	0.6	0.8
libquantum	1.61	33.0	33.1	47.7
h264avc	0.80	8.8	1.6	0.2
omnetpp	2.94	30.9	27.7	29.8
astar	1.79	16.3	9.2	8.2
xalancbmk	2.70	38.0	15.8	11.4
Median	1.35	13.6	7.5	5.4

Typical Memory System Parameters

Feature	Typical values for L1 caches	Typical values for L2 caches	Typical values for paged memory	Typical values for a TLB
Total size in blocks	250–2000	15,000–50,000	16,000–250,000	40–1024
Total size in kilobytes	16–64	500–4000	1,000,000–1,000,000,000	0.25–16
Block size in bytes	16–64	64–128	4000–64,000	4–32
Miss penalty in clocks	10–25	100–1000	10,000,000–100,000,000	10–1000
Miss rates (global for L2)	2%–5%	0.1%–2%	0.00001%–0.0001%	0.01%–2%

Summary

- Cache size is Data + Management (tags, valid, dirty, etc. bits)
- Write misses trickier to implement than reads
- Cache Performance Equations:
  - CPU time = IC × CPI_{stall} × CC
    = IC × (CPI_{ideal} + Memory-stall cycles) × CC
  - AMAT = Time for a hit + Miss rate x Miss penalty